ГЕНЕРАТОРЫ ПСЕВДОСЛУЧАЙНЫХ ЧИСЕЛ

Слеповичев И.И.

май 21

2017

Пособие содержит описание основных методов генерации последовательностей псевдослучайных чисел. Рассмотрены проблемы криптографической стойкости, качества и производительности генераторов псевдослучайных чисел. В курсе дается обзор статистических критериев «случайности» последовательностей чисел.

Учебное пособие

ВВЕДЕНИЕ 3

1. АРИФМЕТИКА В КОНЕЧНЫХ ПОЛЯХ 7

2. СТРУКТУРА ГЕНЕРАТОРА ПСЕВДОСЛУЧАЙНЫХ ЧИСЕЛ 17

3. АЛГОРИТМЫ ГЕНЕРАЦИИ ПСЕВДОСЛУЧАЙНЫХ ЧИСЕЛ 29

4. КРИПТОГРАФИЧЕСКИ СТОЙКИЕ ГПСЧ 59

5. ТЕСТИРОВАНИЕ СТАТИСТИЧЕСКИХ СВОЙСТВ 85

6. ПРЕОБРАЗОВАНИЕ ПСЧ К НУЖНОМУ РАСПРЕДЕЛЕНИЮ 105

СПИСОК ИСТОЧНИКОВ 113

Введение

Случайные числа используются давно и повсеместно. Перечислим некоторые области их применения:

1. Социологические и научные исследования. Подготовка случайных выборок при сборе данных, опросе мнений или в исследовании физических явлений со случайным выбором результатов экспериментов.
2. Моделирование. В компьютерном моделировании физических явлений. Кроме того, математическое моделирование использует случайные числа как один из инструментов численного анализа.
3. Криптография и информационная безопасность. Случайные числа могут использоваться в тестировании корректности или эффективности алгоритмов и программ. Многие алгоритмы используют генерацию псевдослучайных чисел для решения прикладных задач (например, криптографические алгоритмы шифрования, генерация уникальных идентификаторов и др.).
4. Принятие решений в автоматизированных экспертных системах. Использование случайных чисел является частью стратегий принятия решений. Например, для беспристрастности выбора экзаменационного билета студентом на экзамене. Случайность также используется в теории матричных игр.
5. Оптимизация функциональных зависимостей. Некоторые математические методы оптимизации используют стохастические методы для поиска экстремумов функций.
6. Развлечения и игры. Случайность в играх имеет значительную роль. В компьютерных или настольных играх случайность помогает разнообразить игровой процесс.

При более глубоком знакомстве с природой случайности и случайных чисел возникают вопросы о том, что такое «истинно» случайное число, какой должна быть последовательность случайных чисел, каким может быть распределение случайных чисел на некотором интервале (например, временном). В данной книге приведены устоявшиеся формальные определения и популярные алгоритмы, в которых содержатся ответы на данные вопросы.

Необходимость использовать случайные числа в научной работе возникла давно. Сначала для этой цели использовалась урна с шарами, из которой наугад вытаскивали шар с цифрой. Позже были построены механические генераторы случайных чисел. С появлением электронных схем появились и электронные генераторы случайных чисел. Один из первых таких генераторов, предложенный А.М. Тьюрингом, использовал резисторный генератор шума для получения 20 случайных бит поступавших на сумматор [46]. Однако генераторы случайных чисел не всегда давали «хорошие» результаты (понятие «хороших» случайных чисел будет рассмотрено далее). Кроме того, аппаратные генераторы случайных чисел нередко давали сбои. Таблицы же «хороших» случайных чисел, вычисленных заранее, были крайне неудобны в использовании в связи с ограниченностью компьютерной памяти.

В 90-е годы 20 века рост возможностей компьютера, увеличение плотности записи на магнитных и оптических носителях позволило составить достаточно большие таблицы случайно сгенерированных байтов. Так, например, Джордж Марсалья создал большой каталог таблиц случайных чисел объемом 650 мегабайт, что позволило разместить его на оптическом диске.

Однако ни табличный метод, ни аппаратная генерация случайных чисел не могли удовлетворить потребность в надежных, быстрых и эффективных генераторах случайных чисел из-за присущих этим методам «врожденных» недостатков. Поэтому уже на заре компьютерной эры внимание математиков было обращено на алгоритмические способы получения случайных чисел. Так, еще в 1946 году, Джон фон Нейман предложил арифметический способ создания числа, «похожего на случайное». Идея метода заключается в получении следующего случайного числа из предыдущего путем возведения его в квадрат и выделения средних цифр. Например, если предыдущее число было 259, то возведя его в квадрат, мы получим 67081, а средние цифры – 708 – будут очередным случайным числом.

Очевидно, что полученная в результате таких операций последовательность не является случайной, так как полностью определяется формулой и начальным числом. Но во многих приложениях такая псевдослучайная последовательность вполне достаточна, так как выглядит «вполне» случайной. В литературе числа, сгенерированные арифметическим способом, называются *псев-*

дослучайными, а метод их генерации называют *генератором псевдослучайных чисел*.

Последовательности псевдослучайных чисел обладают рядом существенных недостатков. Один из главных недостатков – цикличность последовательности. То есть генератор псевдослучайных чисел может дать, например, 6100 чисел, отличающихся друг от друга, но потом, начиная с 6101-го числа, будет циклично воспроизводить эти 6100 предыдущих чисел. Другим недостатком, присущим всем алгоритмическим методам генерации чисел, является очевидная зависимость последующих чисел от предыдущих. Это может быть незаметно для человеческого восприятия, но впоследствии негативно проявится в момент применения таких чисел при решении прикладной задачи. Алгоритмические методы генерации могут лишь частично «замаскировать» такую зависимость, но не могут избавиться от неё.

Проблема создания быстрого, эффективного генератора псевдослучайных чисел остро стоит и сейчас. Создать генератор, лишенный традиционных недостатков, оказалось очень сложной задачей.

Данная книга является введением в теорию построения и применения генераторов псевдослучайных чисел. В книге опущены математические обоснования описываемых алгоритмов, однако интересующийся выкладками может обратиться к академической литературе, приведенной в списке источников. В книге рассматриваются популярные генераторы псевдослучайных чисел, их свойства, а также методы оценки их эффективности. С другой стороны, структура и содержание книги позволяют использовать её как справочное пособие по алгоритмам и методам генерации псевдослучайных чисел.

Первый раздел содержит базовые сведения из алгебры, лежащие в основе построения рекуррентных последовательностей в конечных полях. Второй раздел посвящен основным понятиям и принципам построения генераторов псевдослучайных чисел. Третий раздел содержит обзор наиболее популярных алгоритмов, не обладающих свойством криптографической стойкости. Четвертый раздел затрагивает вопросы создания криптографически стойких генераторов. В пятом разделе приведены методы оценки статистических свойств и критерии оценки качества генераторов псевдослучайных чисел. Шестой раздел посвящен алгоритмам преобразования массивов чисел к заданному распределению и имеет справочный характер.

Книга представляет интерес для студентов университетов и высших технических учебных заведений, связанных с прикладной криптографией и математическим моделированием.

1. Арифметика в конечных полях

Устройства и алгоритмы, которые будут рассмотрены в данной книге, описываются операциями над элементами в *конечных полях*. Структура и свойства конечных полей определяют свойства последовательностей чисел, вычисленных по рекуррентной формуле. В данном разделе будут рассмотрены основные понятия теории конечных полей. Более подробно с алгебраической теорией конечных полей читатель может ознакомиться в книгах [51, 52, 53].

1.1. Конечное поле

Определение 1.1. *Полем* называется множество с нулем и единицей, в котором определена аддитивная операция + (*сумма*) и мультипликативная операция * (*произведение*) для любых двух элементов. Операции должны подчиняться аксиомам:

1. Коммутативность сложения: $a + b = b + a$.
2. Ассоциативность сложения: $(a + b) + c = a + (b + c)$.
3. Существование обратного элемента по сложению: для любого a существует $–a$ такое, что $a + (-a) = 0$.
4. Коммутативность умножения: $a * b = b * a$.
5. Ассоциативность умножения: $(a * b) * c = a * (b * c)$.
6. Дистрибутивность умножения и сложения: $(a + b) * c = a * c + b * c$.
7. Существование обратного элемента по умножению: для любого a существует a^{-1}, что $a * a^{-1} = 1$.

Все ненулевые элементы конечного поля могут быть представлены как степени некоторого фиксированного элемента поля. Число элементов поля называется *порядком* поля. Поле с конечным порядком называется *конечным полем* или *полем Галуа* и обозначаются $GF(p)$, где p – порядок поля.

Пример. Пусть p – простое число[1]. Тогда множество $\{0,1,2, \dots, p-1\}$ с операциями сложения и умножения по модулю p образует поле $GF(p)$. Другими словами, сумма (произведение) определяется как остаток от деления на p соответ-

[1] Простым называется натуральное число, большее единицы, которое делится без остатка только на 1 и на само себя.

ствующей суммы (произведения). Утверждение $a = b$ в $GF(p)$ означает, что $a - b$ делится на p. В этом случае также говорят, что *a сравнимо* с b по модулю p и записывают $a \equiv b \pmod p$ или $a = b \pmod p$.

Из свойств операций поля следует, что если поле содержит элемент a, то оно должно содержать и все степени $a, a^2, a^3, \ldots$. Наличие обратных элементов означает, что поле также содержит элементы $a^{-1}, a^{-2}, a^{-3}, \ldots$. Вычисляя последовательно различные степени элемента a в конечном поле, мы обязательно встретим элемент a^m, который совпадает с некоторым a^k, вычисленным ранее: $a^m = a^k, k < m$. Умножая обе части этого равенства на a^{-k} получим $a^{m-k} = 1$.

Определение 1.2. Наименьшее из положительных чисел n, для которых $a^n = 1$, называется *порядком элемента a*.

Из определения порядка элемента непосредственно следует, что все элементы $1, a, a^2, \ldots, a^{n-1}$ различны. Иначе, для двух одинаковых элементов $a^m = a^k,\ 0 \le m < k < n$ существовал бы элемент $a^{k-m} = 1$, при $0 < k - m < n$, что противоречит определению. Особым случаем является элемент 0, для которого порядок считается неопределенным.

Запишем несколько важных утверждений, доказательства которых приведены в [51, с. 97-98].

Теорема 1.1. Если a – элемент конечного поля порядка n, то $a^m = 1$ тогда и только тогда, когда m кратно n.

Теорема 1.2. Если порядок элемента a равен n, а порядок элемента b равен m, и наибольший общий делитель m и n равен 1, то порядок $a*b$ равен $m*n$.

Определение 1.3. Будем называть элемент *a примитивным корнем n*-й степени из единицы, если порядок a равен n. В поле порядка q элемент a называется *примитивным элементом поля*, если порядок a равен $q - 1$.

Теорема 1.3. Конечное поле порядка q содержит примитивный элемент порядка $q - 1$, степени которого пробегают все ненулевые элементы поля.

Одним из следствий этой теоремы является теорема, обобщающая теорему Ферма:

Теорема 1.4. Каждый элемент поля порядка q удовлетворяет уравнению

$$x^q - x = 0.$$

Пример. Рассмотрим поле $GF(3) = (\{0,1,2\}, +, *)$, в котором сложение и умножение чисел определены по модулю 3. Например, $2 + 2 = 1 (\mathrm{mod}\, 3)$, а $2 * 2 = 1 (\mathrm{mod}\, 3)$. Степени 2: $2^0 = 1, 2^1 = 2, 2^2 = 1$.

Отметим, что если порядок поля равен некоторому числу $L = p^n$, то поле не может быть образовано из последовательности целых чисел от 0 до $L - 1$, так как не все из этих чисел будут иметь обратный элемент. Например, в множестве $\{0,1,2,3\}$ с операцией сложения и умножения по модулю 4, элемент 2 не имеет обратного, так как $2 * 2 = 0$.

В то же время, для любого простого числа p и натурального $n > 1$, мы можем построить поле порядка p^n, однако структура этого поля будет отличной от структуры поля целых чисел по модулю p^n.

1.2. Поля Галуа $GF(p^n)$

Рассмотрим множество многочленов с коэффициентами из $GF(p)$ (*над полем* $GF(p)$), степени не более $n - 1$. Напомним, что *многочленом* степени m от неизвестной x над полем $GF(p)$ называется выражение вида $a_0 + a_1 x + \cdots + a_m x^m, a_i \in GF(p), a_m \neq 0$. Будем обозначать этот многочлен $a(x)$. Отметим, что данное выражение не рассматривается как функция[1], отображающая $GF(p)$ в $GF(p)$.

Многочлен однозначно определяется вектором своих коэффициентов $(a_0, a_1, \dots, a_m)$. Многочлены можно суммировать, суммируя коэффициенты при одинаковых степенях (с приведением подобных членов по модулю p).

Произведение многочленов $a(x), b(x)$ определяется формулой

$$c(x) = c_0 + c_1 x + \cdots + c_{m+r} x^{m+r},$$
$$c_k = \sum_{i+j=k} a_i b_j.$$

Легко проверить, что множество многочленов над полем $GF(p)$ с операциями сложения, умножения образует коммутативное кольцо $R[x]$, то есть множество с нулем, в котором сложение и умножение имеют свойства коммутатив-

[1] Вопросы интерпретации многочленов как полиномиальных функций рассмотрены в [53, с.325-328].

ности, ассоциативности и связаны законом дистрибутивности умножения относительно сложения[1].

Операция деления с остатком может быть определена для многочленов аналогично делению целых чисел с остатком [53, с. 314]. То есть если $a(x), b(x)$ – многочлены с коэффициентами из $GF(p)$, то существуют однозначно определенные многочлены $q(x)$, $r(x)$, такие, что: либо $r(x) = 0$, либо $\deg r(x) < \deg b(x)$ и $a(x) = b(x)q(x) + r(x)$.

Результат произведения многочленов может иметь степень больше $n - 1$, так как $\deg a(x)b(x) = \deg a(x) + \deg b(x)$. Поэтому, для выполнения условия $\deg a(x)b(x) \leq n - 1$ нам понадобится дополнительное действие. Таким действием мы выберем процедуру взятия остатка от деления $a(x)b(x)$ на некоторый многочлен $g(x)$. В качестве многочлена $g(x)$ можно взять так называемый *неприводимый многочлен*.

Определение 1.4. Многочлен $g(x)$ над полем $GF(p)$ называется *неприводимым* над этим полем, если из того, что $g(x) = a(x)b(x)$, следует, что либо $a(x)$, либо $b(x)$ является многочленом нулевой степени (то есть константой). Если же $a(x)$ и $b(x)$ не константы, то $g(x)$ называется *приводимым*.

Неприводимый многочлен играет для поля порядка p^n ту же роль, что и простое число p для поля $GF(p)$. Мы можем использовать его для определения операции умножения в кольце многочленов следующим образом: результатом будем считать остаток от деления $a(x)b(x)$ на неприводимый многочлен $g(x)$:

$$c(x) = a(x)b(x) \bmod g(x).$$

Мы определили операцию сложения и умножения, результат применения которых снова принадлежит множеству многочленов степени не более $n - 1$. Эти операции, как следует из их определения, ассоциативны, коммутативны и дистрибутивны, а по сложению каждый многочлен имеет обратный элемент. То есть данное множество многочленов с операциями сложения и умножения образует коммутативное кольцо, которое мы будем обозначать $R[x]/(g)$.

Кольцо $R[x]/(g)$ можно рассматривать как кольцо классов вычетов многочленов по модулю неприводимого многочлена $g(x)$. Два многочлена будем считать эквивалентными в том и только в том случае, когда у них одинаковые

[1] Эти свойства в определении конечного поля перечислены под номерами 1-6.

остатки при делении на $g(x)$. При таком определении количество элементов в $R[x]/(g)$ равно количеству различных остатков при делении на многочлен $g(x)$.

Остаток $r(x)$ при делении на $g(x)$ имеет вид

$$r(x) = \sum_{i=0}^{n-1} a_i x^i,$$

где $a_i \in GF(p)$. Число возможных комбинаций этих коэффициентов дает нам все возможные остатки при делении на многочлен $g(x)$ и равно p^n.

Можно также показать, что при условии неприводимости многочлена $g(x)$, каждый элемент $a(x) \in R[x]/(g)$ имеет обратный по умножению – $a^{-1}(x) \in R[x]/(g)$. Это означает, что верна

Теорема 1.5. Пусть $R[x]$ – кольцо многочленов над полем $GF(p)$, где p – простое число, и многочлен $g(x) \in R[x]$. Фактор-кольцо $R[x]/(g)$ кольца $R[x]$ по модулю главного идеала[1] (g) является полем тогда и только тогда, когда $g(x)$ – неприводимый многочлен в кольце $R[x]$.

Не менее важен вопрос о том, при каких условиях для некоторого простого числа p и натурального n существует конечное поле порядка p^n. Рассмотрим некоторые следствия из теоремы 1.3.

Теорема 1.6. Любой элемент x поля $G = GF(p^n)$ удовлетворяет уравнению $x^q = x$, где $q = p^n$.

Отметим также, что в любом коммутативном кольце R, $(x - a)$ делит многочлен $p(x) \in R[x], a \in R$ в том и только в том случае, когда $p(a) = 0$ в R.

Из этого, а также из теоремы 1.6 следует, что для любого $x_i \in R[x]/(g)$ выполняется: $(x - x_i)$ делит $(x^q - x)$, где $q = p^n$ и, следовательно

$$x^q - x = \prod_{x_i \in G} (x - x_i).$$

Этот многочлен примечателен тем, что он не имеет кратных корней в поле $GF(p)$, и, следовательно, раскладывается в этом поле на p^n линейных множителей.

---

[1] Главный идеал (g) кольца $R[x]$ – подкольцо, замкнутое относительно умножения $g \in R[x]$ на любой элемент этого кольца.

Так как конечное поле из p элементов существует для каждого простого числа p, то при наличии неприводимого многочлена над этим полем, можно построить конечное поле из $q = p^n$ элементов. При этом доказано, что для любого $n \geq 2$ и любого простого p, в кольце многочленов $R[x]$ найдется хотя бы один неприводимый нормированный многочлен[1] над этим полем. Таким многочленом может быть, например, $x^q - x$. Сформулируем этот результат в виде теоремы.

Теорема 1.7. Для каждого простого числа p и каждого натурального $n \geq 2$ существует конечное поле из p^n элементов (единственное с точностью до изоморфизма), в качестве которого мы можем взять фактор-кольцо $R[x]/(g)$ по модулю главного идеала по неприводимому в этом кольце многочлену $g(x)$. Это поле называется полем Галуа порядка p^n и обозначается $GF(p^n)$.

Полное доказательство этой теоремы приведено в [51, с. 111].

Для построения поля $GF(p^n)$ мы можем использовать так называемый *примитивный многочлен*.

Определение 1.5. Многочлен $g(x)$ степени n над полем $GF(p)$ называется *примитивным*, если он не делит нацело ни один многочлен вида $x^s - 1$, где $s < p^n - 1$.

Такие многочлены являются неприводимыми. Кроме того, эти многочлены – в точности корни уравнения $x^{q-1} = 1$ в поле $GF(q)$, где $q = p^n$. Каждый из этих многочленов порождает поле $GF(p^n)$ над $GF(p)$.

Теорема 1.8. В $GF(p)$ имеется $\phi(p^n - 1)/n$ многочленов степени n со старшим коэффициентом равным единице, корни которых имеют период $p^n - 1$. Здесь $\phi(k)$ – функция Эйлера, то есть число натуральных чисел меньших k и взаимно простых с k.

Пример. Пусть $p = 2,\ n = 4,\ c(x) = x^4 + x + 1$ – примитивный над $GF(2)$. Тогда поле $GF(2^4)$ состоит из многочленов: $0,\ 1,\ x,\ x + 1,\ x^2 + x + 1,\ x^3 + x^2 + x + 1$. Приведем примеры выражений с этими многочленами (отдельный многочлен будем выделять скобками):

$$1 + (x + 1) = x,$$
$$x + (x + 1) = 1,$$

[1] Нормированный многочлен – многочлен, коэффициент при старшей степени которого равен единице.

$(x^3 + x^2 + 1) + (x^2 + x + 1) = x^3 + x,$
$(x + 1)(x^2 + x) = x^3 + x,$
$(x^2 + 1)(x^3 + x^2 + x) = x^5 + x^4 + x^2 + x \left(mod\ (x^4 + x + 1)\right) = 1 + x.$

1.3. Расширения полей. Векторное пространство многочленов

Пусть F и G – два поля. Говорят, что G является *расширением* поля F, если $F \subset G$. Например, поле комплексных чисел является расширением поля действительных чисел, а поле действительных чисел является расширением поля рациональных.

Любое расширение G поля F можно рассматривать как векторное пространство над F. Это означает, что G – коммутативная группа по операции сложения (т.е. для неё выполняются аксиомы 1-3) и что определено умножение её элементов на элементы F (скаляры), обладающее свойствами:

$$a(b\gamma) = (ab)\gamma,$$
$$(a + b)\gamma = a\gamma + b\gamma,$$
$$a(\gamma + \delta) = a\gamma + a\delta,$$
$$1\gamma = \gamma$$

для всех $a, b \in F$ и $\gamma, \delta \in G$. В поле G, как в векторном пространстве, можно выбрать некоторый базис – множество векторов $\beta_1, \dots, \beta_n$, что любой $\alpha \in G$ однозначно представим в виде линейной комбинации $\alpha = x_1\beta_1 + \cdots + x_n\beta_n$. Кроме того, можно показать, что в n-мерном векторном пространстве любые $n + 1$ векторов $\alpha_0, \alpha_1, \dots, \alpha_n \in G$ линейно зависимы, то есть для них выполняется равенство $c_0\alpha_0 + \cdots + c_n\alpha_n = 0$, и не все $c_i \in F$ равны нулю.

Например, коммутативное кольцо всех многочленов $R[x]$ над некоторым полем F является бесконечномерным векторным пространством. А коммутативное кольцо многочленов степени $\leq n$ над полем F образует $(n + 1)$-мерное пространство с базисом $1, x, x^2, \dots, x^n$. Таким кольцом может быть, например, фактор-кольцо $R[x]/(x^{n+1} - 1)$.

Размерность векторного пространства G, являющегося расширением поля F, называется *степенью* этого расширения.

Определение 1.6. Будем говорить, что поле G является *простым алгебраическим расширением* поля $GF(p)$, если оно получено из поля $GF(p)$ присоединением корня неприводимого над $GF(p)$ многочлена $g(x)$.

Если c – такой корень, то все элементы поля имеют вид

$$a_{n-1}c^{n-1} + \cdots + a_1 c + a_0,$$

где коэффициенты $a_{n-1}, \ldots, a_1, a_0$ пробегают все возможные значения из поля $GF(p)$.

1.4. Вычисления обратного элемента в $R[x]/(g)$

Выше в п. 1.2 было описано, как вычислять сумму и произведение многочленов. Вычисление обратного элемента в $GF(p^n) = R[x]/(g)$ несколько сложнее. В общем случае, если ненулевой элемент поля $GF(p^n)$ представлен многочленом $a(x)$, то $a(x)$ должен быть взаимно прост с $g(x)$. Алгоритмом Евклида можно найти многочлены $s(x)$ и $t(x)$, такие, что

$$s(x)a(x) + t(x)g(x) = 1.$$

Так как $g(x) = 0$ в G, элемент, обратный многочлену $a(x)$, равен многочлену $s(x)$.

1.5. Вычисления в $GF(2^n)$.

Рассмотрим вычисления в поле $G = GF(2^n)$. Как было сказано выше, это поле мы можем рассматривать как векторное пространство над $Z_2 = \{0,1\}$. В качестве базиса можно выбрать элементы $1, c, \ldots, c^{n-1}$, где c – корень неприводимого многочлена степени n над Z_2. В таком поле каждый многочлен отождествляется с двоичным вектором длины n. Сложение определяется покоординатным сложением элементов (в Z_2), а умножение $a(x)$ на $b(x)$ определяется как $a(x)b(x) (\mathrm{mod}\, g(x))$. Если $g(x) \in R[x]$ – примитивный, то мы можем выбрать корень этого многочлена в качестве элемента поля G, степени которого пробегают все элементы поля G. Все вычисления над элементами поля мы можем выполнять как вычисления над соответствующими степенями примитивного элемента поля – корня многочлена $g(x)$.

Например, для примитивного многочлена $g(x) = x^4 + x + 1$, примитивным корнем будет $\omega = x$ или 0010 (в виде двоичного слова). Сопоставление степеней этого элемента с двоичными векторами представлено в таблице 1.1.

Таблица 1.1. Различные представления многочленов в поле $GF(2^4)$.

Вектор $a_3a_2a_1a_0$	**Многочлен**	**Степень ω**
0001	1	1
0010	x	ω
0100	x^2	ω^2
1000	x^3	ω^3
0011	$x+1$	ω^4
0110	x^2+x	ω^5
1100	x^3+x^2	ω^6
1011	x^3+x+1	ω^7
0101	x^2+1	ω^8
1010	x^3+x	ω^9
0111	x^2+x+1	ω^{10}
1110	x^3+x^2+x	ω^{11}
1111	x^3+x^2+x+1	ω^{12}
1101	x^3+x^2+1	ω^{13}
1001	x^3+1	ω^{14}

Опираясь на формулу умножения $\omega^s\omega^r = \omega^{(s+r) \bmod q}$, где $q = 2^n - 1$, $\omega^s \equiv a(x), \omega^r \equiv b(x)$, мы можем осуществлять эффективные вычисления, не используя затратных алгоритмов умножения и деления многочленов.

Например, $(x^2+x+1)(x+1) = \omega^{10}\omega^4 = \omega^{14} = x^3+1$. То же самое можно было получить, используя операции умножения с приведением по модулю многочлена x^4+x+1. Аналогично можно вычислять результат деления многочленов: $(x^3+x^2+1)/(x^2+1) = \omega^{13}/\omega^8 = \omega^5 = x^2+x$.

Такой способ вычисления в поле называется табличным. Он позволяет эффективно вычислять не только произведение и сумму элементов поля, но и

обратный элемент, используя равенство $\omega^{2^n-1} = 1$. Так, например, для вычисления обратного элемента к $x^3 + x = \omega^9$ в $GF(2^4)$ мы воспользуемся равенством $\omega^{15} = \omega^9\omega^6 = 1$. Из чего получаем, что $\omega^6 \equiv x^3 + x^2$ является обратным к многочлену $x^3 + x$.

2. Структура генератора псевдослучайных чисел

2.1. Основные понятия

Сначала приведем определения основных понятий теории генерации случайных чисел.

Определение 2.1. *Случайное число* – число, представляющее собой реализацию случайной величины [6].

Определение 2.2. *Детерминированный алгоритм* – алгоритм, который возвращает те же выходные значения при тех же входных значениях.

Определение 2.3. *Псевдослучайное число* – число, полученное детерминированным алгоритмом, используемое в качестве случайного числа.

Определение 2.4. *Физическое случайное число (истинно случайное)* – случайное число, полученное на основе некоторого физического явления.

Как правило, генерация случайного числа состоит из двух этапов:

1. генерация нормализованного случайного числа (то есть равномерно распределенного от 0 до 1);
2. преобразование нормализованных случайных чисел r_i в случайные числа x_i, которые распределены по заданному закону распределения или в необходимом интервале.

Определение 2.5. *Генератор случайных бит* (ГСБ) — это устройство или алгоритм, который выдает последовательность статистически независимых и несмещенных бит (то есть подчиняющихся закону распределения) [3, §§5.1.1].

Замечание. Генератор случайных бит может быть использован для генерации равномерно распределенных случайных чисел. Например, случайное целое число в интервале $[0;\ n]$ может быть получено из сгенерированной последовательности случайных бит длины $\lfloor \lg n \rfloor + 1$ путем конвертации её в соответствующую систему исчисления. Если полученное в результате целое число превосходит n, то его можно отбросить и сгенерировать еще одну последовательность бит. Поэтому далее мы будем использовать *термин генератор случайных чисел* наравне с термином *генератор случайных бит*.

Определение 2.6. *Генератором псевдослучайных бит (детерминированным ГПСБ)* будем называть детерминированный алгоритм[1], который получает на вход двоичную последовательность длины k и выдает на выходе двоичную последовательность длины $l \gg k$ (l значительно больше k), которая «выглядит случайной». Входное значение ГПСБ называется *начальным* вектором (также называют инициализационным вектором и обозначают *IV*), а выход называется псевдослучайной последовательностью бит.

Поясним понятие «выглядит случайной». Понятно, что последовательность, сгенерированная детерминированным алгоритмом, не является случайной. Однако цель алгоритма в том, чтобы взять некоторую маленькую последовательность истинно случайных чисел и использовать её для генерации длинной последовательности, не отличимой от истинно случайной последовательности чисел той же длины. Убедиться в том, что последовательность чисел случайна (или не случайна) можно либо при помощи статистических тестов, выявляющих специфические особенности случайных последовательностей, либо аналитико-вычислительными методами. Подробно о тестах речь пойдет в пятом разделе книги.

Определение 2.7. Говорят, что ГПСБ проходит все полиномиальные по времени вероятностные тесты на статистическую случайность, если не существует полиномиального по времени[2] вероятностного алгоритма, который бы мог корректно отличить выходную последовательность генератора от истинно случайной последовательности той же длины с вероятностью превышающей $1 / 2$.

Определение 2.8. Говорят, что ГПСБ успешно проходит тест на следующий бит, если не существует полиномиального по времени алгоритма, который может по входным l битам последовательности s предсказать $(l + 1)$-й бит s с вероятностью превышающей $1 / 2$.

Более формально, ГПСБ проходит тест на следующий бит, если для любого $i \in \mathbb{N}$ и любого вероятностного полиномиального по времени алгоритма $A: \{0,1\}^i \to \{0,1\}$, выполняется следующее неравенство:

---

[1] В общем случае мы можем говорить о псевдослучайной функции. Но в данной работе нас, прежде всего, интересуют аспекты реализации вычислимых функций на компьютере, поэтому здесь и далее речь будет идти именно об алгоритме.

[2] То есть время выполнения теста ограничено сверху значением полинома, вычисленного от длины l выходной последовательности.

$$\left|P\left(A\left(s_1^{i-1}\right)=s_i\right)-\frac{1}{2}\right|<O\left(v(n)\right),$$

где $O\left(v(n)\right)$ - обозначение функции, убывающей быстрее, чем обратный полином степени n.

Несмотря на то, что определение 2.7 накладывает более строгие условия на ГПСЧ, чем определение 2.8, можно доказать, что эти определения эквивалентны.

Утверждение 2.1. (Универсальность теста на следующий бит) ГПСБ проходит тест на следующий бит тогда и только тогда, когда он проходит все полиномиальные статистические тесты.

Доказательство этого утверждения получил Эндрю Яо в 1982 году [41].

2.2. Виды генераторов случайных чисел

Генераторы случайных чисел по способу получения чисел делятся на:

- аппаратные;
- табличные;
- алгоритмические.

Табличные генераторы в качестве источника случайных чисел используют заранее подготовленные таблицы, содержащие проверенные некоррелированные числа и не являются генераторами в строгом понимании этого понятия. Недостатки такого способа очевидны: использование внешнего ресурса для хранения чисел, ограниченность последовательности, предопределенность значений. В качестве примера табличного метода можно привести книгу [2].

Аппаратные генераторы (истинно) случайных последовательностей должны обладать источником энтропии. Разработка генераторов, использующих источники энтропии, генерирующих не коррелированные и статистически независимые числа – достаточно сложная задача. Кроме того, для большинства криптографических приложений такой ГПСЧ не должен быть предметом изучения и воздействий противной стороны. О криптографически стойких генераторах случайных чисел речь пойдет в четвертом разделе.

Алгоритмический генератор является комбинацией физического генератора и детерминированного алгоритма. Такой генератор использует ограниченный набор данных, полученный с выхода физического генератора для создания

длинной последовательности чисел преобразованиями исходных чисел. Данный вид генераторов представляет наибольший интерес в силу его очевидных преимуществ над генераторами случайных чисел других видов. Разбор свойств, достоинств и недостатков алгоритмических ГПСЧ является основной темой данной книги.

2.3. Стандарты и нормативные документы

Прежде чем перейти к рассмотрению вопросов генерации случайных чисел, сделаем небольшой обзор ресурсов, которые можно использовать в качестве источников методической информации по генерации ПСЧ. На данное время существует большое количество статей, публикаций, книг и обзоров по методам защиты информации. Однако далеко не все публикации проходят верификацию и исследование на предмет качества изложенного материала и безопасности описываемых алгоритмов. Более того, часто авторы даже не упоминают о том, что изложенный в публикации алгоритм был скомпрометирован или, по крайней мере, предполагает наличие некоторых уязвимостей при его использовании. Все это в равной степени относится и к методам генерации псевдослучайных чисел. Поэтому следует крайне осторожно относиться к источникам, в которых излагается тот или иной алгоритм генерации ПСЧ. Ниже приведены некоторые информационные ресурсы, имеющие достаточно высокую степень надежности, которые могут быть использованы как в ознакомительных, так и в прикладных целях.

2.3.1. Международные действующие стандарты ISO/IEC

ISO[1] – международная организация по стандартизации. Это независимая, неправительственная организация и самый крупный в мире разработчик международных стандартов. Существует с 1947 года и опубликовала десятки тысяч документов описывающих различные стандарты. Для нас, прежде всего, интересны следующие стандарты:

- ISO/IEC 10116-91 – «Банковское дело. Режимы работы n-бит блочного алгоритма шифрования».
- ISO/IEC 10118-1,2-88 – «Информационные технологии. Шифрование данных. Хэш-функция для цифровой подписи».

[1] Официальный сайт организации – http://www.iso.org.

- ISO/IEC CD 10118-3,4 – «Информационные технологии. Защита информации. Функции хеширования».
- ISO/IEC CD 14888 – «Информационные технологии. Защита информации. Цифровая подпись с добавлением».

Широкое внедрение алгоритмов, описанных в этих стандартах, представляется малореальным, поскольку политика крупных государств направлена, как правило, на использование собственных криптографических алгоритмов.

2.3.2. Действующие стандарты Российской Федерации, касающиеся генерации ПСЧ

Основная в России организация, занимающаяся стандартизацией – это Росстандарт[1], – федеральный орган исполнительной власти, осуществляющий функции по оказанию государственных услуг в области регулирования стандартизации, экспертизы национальных стандартов, технических требований, предписаний и т.п. Совместно с институтами, комиссиями от федеральных ведомств и министерств Росстандарт принимает национальные российские стандарты. В частности, ГОСТ Р 34.11-2012 был разработан Центром защиты информации и специальной связи ФСБ России с участием ОАО «ИнфоТеКС».

- ГОСТ Р 28147-89 – «Системы обработки информации. Защита криптографическая. Алгоритм криптографического преобразования». Этот стандарт описывает блочный шифроалгоритм, который может быть использован в качестве компонента ГПСЧ.
- ГОСТ Р 34.11-2012 – «Информационная технология. Криптографическая защита информации. Функция хеширования». Данный стандарт описывает метод формирования функции хэширования, который может быть использован в качестве компонента ГПСЧ.
- ГОСТ Р ИСО 28640-2012 – «Статистические методы. Генерация случайных чисел». Стандарт описывает методы генерации случайных чисел, подчиняющихся равномерному и другим законам распределения, используемых при применении метода Монте-Карло. В этот стандарт не включены криптографические методы генерации случайных чисел.

---

[1] Официальный сайт организации – http://www.gost.ru.

2.3.3. Публикации NIST

Отдельного обзора заслуживают публикации NIST[1] – американского национального института стандартизации и технологий. В составе института функционирует компетентный и имеющий серьезный вес в США центр по компьютерной безопасности – CSRC, объединяющий специалистов федеральных служб, университетов, крупнейших ИТ-компаний США. Центр публикует с начала 1990-х годов Стандарты (FIPS) и более детальные разъяснения/рекомендации (Special Publications) в области информационной безопасности. Рекомендациям, созданным CSRC, присваивается код 800.

Из публикаций NIST нас, прежде всего, интересуют:

- SP 800-38A,B – «Recommendation for Block Cipher Modes of Operation». Рекомендации по построению блочного шифроалгоритма, который может быть использован в ГПСЧ.
- SP 800-90A – «Recommendation for Random Number Generation Using Deterministic Random Bit Generators». Рекомендации для генерации случайных чисел с использованием детерминированных алгоритмов генерации случайных битов. Эти рекомендации описывают криптографически стойкие ГПСЧ.
- SP 800-22 – «A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications». Набор статистических тестов для генераторов случайных и псевдослучайных чисел для криптографического применения.

2.3.4. Стандарты ANSI

Кроме того, в США есть Американский национальный институт стандартов – ANSI[2], который занимается преимущественно технологиями торговли и коммуникаций. Для нас интерес представляют стандарты ANSI X9.82, ANSI X9.17, содержащие описание ГПСЧ.

---

[1] Официальный сайт организации – http://www.nist.gov.
[2] Официальный сайт организации – http://www.ansi.org.

2.4. Структура генератора псевдослучайных чисел

Ранее было дано формальное определение генератора псевдослучайных чисел. Построение ГПСЧ на практике является сложной задачей состоящей из решения ряда подзадач. При этом качество решения любой из подзадач оказывает критическое влияние на решение всей задачи в целом, как это не редко бывает в информационной безопасности.

Рассмотрим функциональную модель ГПСЧ. За основу будем брать рекомендации, изложенные в [9].

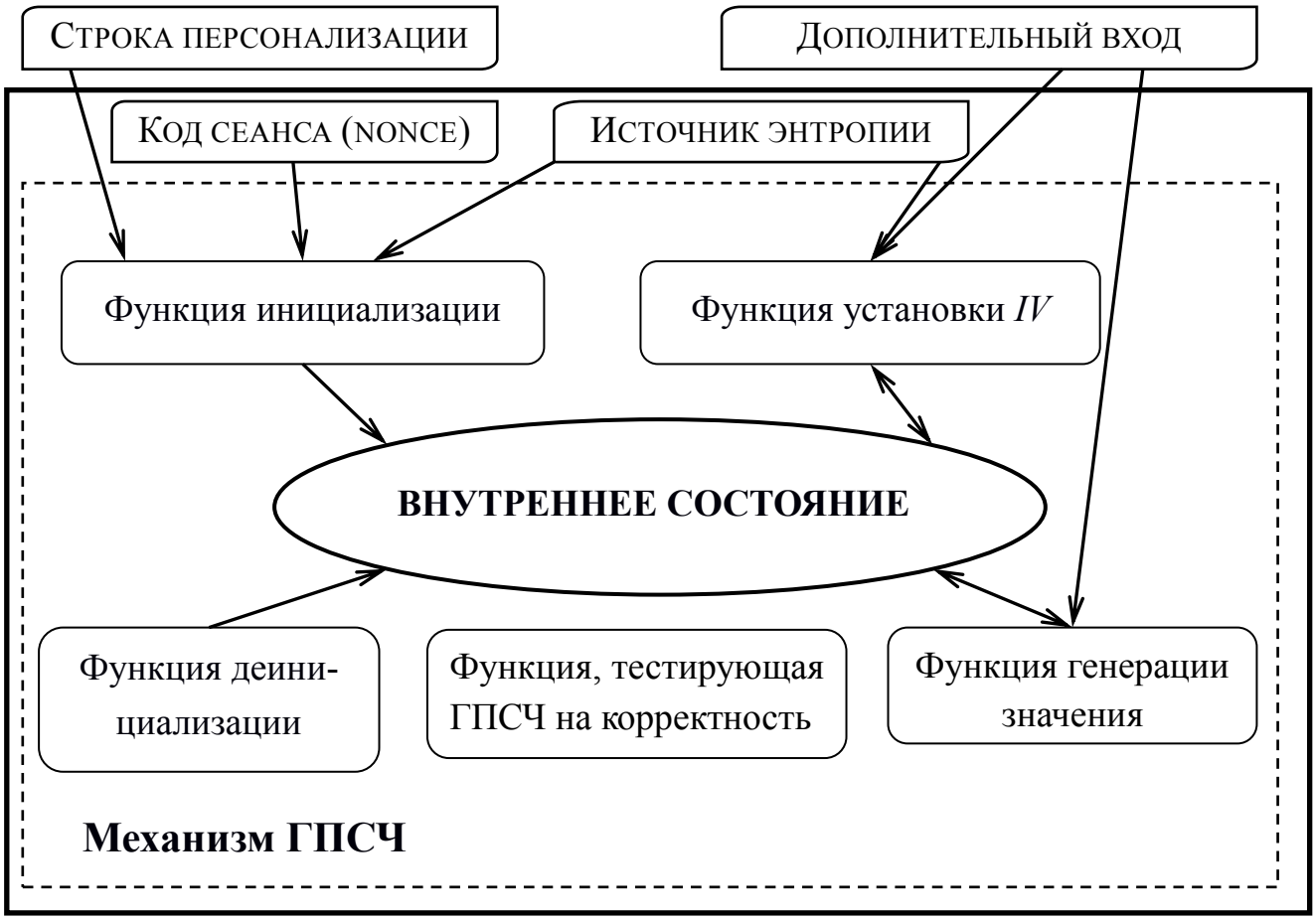

Рисунок 2.1. Функциональная модель ГПСЧ.

Основные компоненты функциональной модели ГПСЧ:

- Источник энтропии;
- Дополнительный вход;
- Строка персонализации;
- Внутреннее состояние;

- Набор внутренних функций: функция инициализации, функция деинициализации, функция тестирующая корректность ГПСЧ, функция генерации значения, функция установки инициализационного вектора.

Рассмотрим подробнее эти компоненты.

2.4.1. Источник энтропии

Источник энтропии (источник случайности) – это механизм, генерирующий физические случайные величины для дальнейшего их использования в качестве инициализационного вектора ГПСЧ. Источник энтропии (аппаратный генератор случайных чисел) и инициализационный вектор, генерируемый для ГПСЧ, должны держаться в секрете и быть надежно защищены. Секретность данной информации является базовым требованием к безопасности ГПСЧ. Если же криптографическая стойкость от ГПСЧ не требуется, источник энтропии должен просто предоставлять случайные числа в нужном количестве. Более подробно об источниках энтропии говорится в п. 2.5.

2.4.2. Дополнительный вход

На этот вход может быть передана некоторая дополнительная информация для генерации случайных чисел. Это может быть как секретная, так и не секретная информация, в зависимости от требований приложения. Как бы то ни было, безопасность ГПСЧ не должна зависеть от секретности этих данных. По возможности, данные с дополнительного входа должны проверяться на корректность. Например, в случае использования значения системного времени, должны проверяться формат и корректность значения времени.

Кроме того, на вход рекомендуется подавать *код сеанса* (nonce) – некоторое значение, уникальное для данного сеанса генерации случайных чисел, которое имеет минимальные шансы быть повторенным в другом сеансе. Оно может быть сформировано из текущего времени (количества тиков), номера сеанса или других величин, не повторяющихся от сеанса к сеансу.

2.4.3. Строка персонализации

В момент инициализации для генерации инициализационного вектора необходимо использовать *строку персонализации*. Суть строки персонализации в том, чтобы сделать текущую инициализацию уникальной относительно всех

остальных инициализаций. Поэтому строка персонализации должна быть настолько уникальной, насколько это возможно. Кроме того, она может включать секретную информацию. В [9] приведены возможные варианты данных, включаемых в строку персонализации:

— серийные номера устройств,
— открытые ключи,
— идентификаторы пользователя,
— временные метки,
— сетевые адреса,
— массивы инициализационных данных,
— специальные ключи, специфичные для ГПСЧ,
— идентификаторы приложения,
— идентификаторы версии протокола,
— случайные числа,
— сеансовый код.

2.4.4. Внутреннее состояние

Внутреннее состояние – это память ГПСЧ, в которой содержатся все параметры, переменные и другие сохраненные значения, необходимые для работы ГПСЧ. В частности, внутреннее состояние хранит текущее значение ПСЧ.

2.4.5. Внутренние функции ГПСЧ

Внутренние функции управляют внутренним состоянием и реализуют ключевые алгоритмы ГПСЧ. В NIST SP 800-90A рекомендуют использовать пять функций, представленных на рисунке 2.1. Рассмотрим их подробнее.

2.4.5.1. Функция инициализации

Функция инициализации – получает число от источника энтропии, комбинирует его со строкой персонализации, текущим временем и создает, таким образом, инициализационный вектор ГПСЧ.

ГПСЧ должен быть инициализирован перед началом генерации псевдослучайных чисел. Функция инициализации делает следующее:

1. проверяет корректность входных параметров,
2. определяет уровень криптостойкости,

3. определяет специфические параметры ГПСЧ (например, набор параметров эллиптической кривой),
4. получает значение от источника энтропии с достаточным уровнем надежности,
5. получает код сеанса (nonce),
6. определяет начальное внутреннее состояние, используя алгоритм инициализации.

2.4.5.2. Функция установки начального вектора (реинициализации)

Функция установки начального вектора получает новое значение от источника энтропии, комбинирует его с текущим внутренним состоянием и значением дополнительного входа и генерирует новый начальный вектор и новое внутреннее состояние для следующего запроса к ГПСЧ. Для краткости будем называть эту функцию *реинициализацией*. Реинициализация добавляет случайности в процесс генерации псевдослучайных чисел и может быть:

- явно запрошена приложением потребителем ПСЧ;
- выполнена в соответствии с правилами защиты от предсказания генерируемых чисел;
- выполнена после определенного числа сгенерированных чисел;
- выполнена по внешнему событию (например, в момент доступности источника энтропии).

В некоторых ГПСЧ нет отдельной функции реинициализации. Вместо этого используется функция инициализации.

Функция реинициализации:

1. Проверяет корректность входных параметров.
2. Получает истинно случайное число от источника энтропии.
3. Используя алгоритм реинициализации, комбинирует текущее внутреннее состояние с новым истинно случайным числом и дополнительным входным значением для определения нового внутреннего состояния.

2.4.5.3. Функция генерации значения

Функция генерации значения – создает псевдослучайное число по запросу, используя текущее внутреннее состояние и меняя внутреннее состояние для следующего запроса. Данная функция выполняет следующие действия:

1. Проверяет корректность входных параметров.
2. Вызывает функцию реинициализации, если это необходимо (см. п. 2.4.5.2).
3. Генерирует число, используя алгоритм генерации ПСЧ.
4. Вычисляет новое внутреннее состояние.

2.4.5.4. Функция деинициализации

Функция деинициализации очищает значения внутреннего состояния. Это необходимо делать в целях противодействия анализу системы.

2.4.5.5. Функция, тестирующая ГПСЧ на корректность

Функция тестирующая ГПСЧ на корректность проводит тест на корректность работы ГПСЧ и сообщает о результате приложению–потребителю ГПСЧ.

Примером реализации ГПСЧ на основе описанных здесь рекомендаций может служить аппаратно-программный ГПСЧ реализованный фирмой Intel на базе архитектуры Intel 64. Этот ГПСЧ реализован в инструкции процессора RdRand (RDRAND) и соответствует стандартам безопасности и криптографическим стандартам NIST SP800-90, FIPS 140-2, и ANSI X9.82 [4].

2.5. Аппаратные генераторы случайных чисел

Определение 2.9. *Аппаратный генератор случайных чисел* – это устройство, использующее для создания случайных чисел замеры параметров некоторых физических процессов. Как правило, аппаратный генератор случайных чисел состоит из источника энтропии и устройства, преобразующего значения, полученные с источника энтропии, в нужный формат.

Источниками энтропии (ИЭ) могут быть:

- подбрасывание монеты;
- временные задержки между моментами излучения частиц в процессе радиоактивного распада;
- тепловые шумы при работе полупроводникового диода или резистора;
- частотные отклонения свободно работающего генератора частот;
- фотоэффект — испускание электронов веществом под действием света;
- звук от микрофона или видео с подключенной камеры;

- состояние некоторых блоков памяти компьютера.

Построение аппаратного генератора случайных чисел трудоёмкая задача, в рамках которой нужно решить такие проблемы как настройка диапазонов фиксируемых случайных величин, оцифровка аналоговых результатов, изоляция физического источника случайности от внешних воздействий и т.п.

Пример. В качестве источника энтропии может быть использован электрический шум диода. В диоде шумовой сигнал достаточно велик вследствие эффекта лавинного нарастания заряда. Поэтому диод часто используют как источник шума. Например, NC2401 фирмы NoiseCom [7]. У этого элемента есть источник шума и встроенный усилитель, ширина полосы частот которого составляет 1 ГГц, а амплитуда – 160 мВ.

Для преобразования шумового сигнала в цифровую форму могут быть использованы следующие методы:

- Аналогово-цифровое преобразование;
- Наблюдение последовательности импульсов с определением количества импульсов в единицу времени;
- Наблюдение последовательности импульсов с определением интервала времени между последовательными импульсами;
- Поскольку у аналогово-цифрового преобразователя могут появляться ошибочные значения, гистограммы значений после преобразования не показывают равномерного распределения. Для получения большей равномерности распределения может быть применена дополнительная обработка значений: два бита конвертируют в один по правилу: $(0,1) \rightarrow 0$, $(1,0) \rightarrow 1$, $(0,0) \rightarrow$не используют, $(1,1) \rightarrow$ не используют.

Рассмотрение аппаратных ГСЧ выходит за рамки круга вопросов данной книги. Интересующиеся этой темой могут обратиться к соответствующим описаниям на сайтах компаний производителей такого оборудования [7].

3. Алгоритмы генерации псевдослучайных чисел

Разработка алгоритмических генераторов может быть еще более сложной задачей по сравнению с созданием аппаратных генераторов случайных чисел. Напомним, что в программных генераторах используются начальные векторы – некоторые истинно случайные числа для генерации всех остальных чисел последовательности. Таким образом, с некоторого источника энтропии снимаются данные, которые конвертируются в начальное истинно случайное число. Далее это число используется в алгоритме для генерации других членов последовательности.

Из-за дороговизны аппаратных генераторов случайных чисел в большинстве случаев, в качестве источника энтропии используются ресурсы вычислительной машины, на которой выполняется программа генерации ПСЧ. При отсутствии аппаратного генератора случайных чисел в качестве источника энтропии могут использоваться:

1. состояние системных часов;
2. время задержек между нажатиями клавиш клавиатуры или движениями мышки;
3. содержимое буферов ввода/вывода;
4. значения, получаемые при работе системы (время загрузки системы, сетевая активность и т. п.).

Генераторы ПСЧ, основанные на преобразованиях системного времени, имеют недостатки: алгоритм, использующий ГПСЧ, может иметь свои «привязки» к системному времени, и, таким образом, сгенерированные в эти моменты числа будут менее случайны. Например, программа должна выдавать серию случайных чисел вначале каждой секунды.

ГПСЧ, использующие в качестве источника энтропии события, генерируемые пользователем (задержки между нажатиями клавиш, координаты движения мыши), при подробном анализе оказываются не равномерно распределенными, поэтому некоторые из чисел менее случайны.

Хороший программный генератор случайных бит должен использовать как можно больше различных источников энтропии. Это уменьшит возможность злоумышленника проанализировать алгоритм создания случайных бит, а также повысит надежность генератора для случаев, когда один или несколько

источников энтропии могут выйти из строя. Каждый из источников энтропии должен быть, по возможности, упрощен. Полученные случайные последовательности «перемешиваются» при помощи некоторой специальной функции. Один из способов такого «перемешивания» – это применение к последовательности функции хэширования.

Рассмотрим способы получения случайных чисел на интервале [0,1]. Понимая, что точность представления действительных чисел в компьютере ограничена вполне определенными величинами, будем использовать тот факт, что любое число заданной точности из интервала [0,1] может быть однозначно преобразовано в число из интервала $[0, m]$. И наоборот, число из интервала $[0, m]$ делением на m преобразуется к интервалу [0,1].

Общая формула для алгоритмического ГПСЧ выглядит так:

$$X_{i+1} = f(X_{i-k+1}, X_{i-k+2}, \dots, X_i),$$

где f – некоторая функция преобразования k последних членов последовательности чисел в новое значение.

Последовательности из таких чисел обязательно образуют циклы. Повторяющиеся циклы называют *периодом*. Другими словами, такие последовательности всегда имеют период. Период последовательности ПСЧ – одно из ключевых её свойств, которое оценивается в первую очередь.

Достоинствами алгоритмических генераторов являются быстродействие, компактность реализации. Основной недостаток – низкое качество «случайности», – такие последовательности, как правило, не проходят большинства полиномиальных тестов на случайность.

3.1. Метод срединных квадратов

Один из первых алгоритмических методов получения равномерно распределенных псевдослучайных чисел получил название "*метод середины квадрата*". Он был предложен Джоном фон Нейманом и заключается в следующем:

1. Выбрать начальное случайное число X_0 имеющее n-разрядное представление (возможно полученное от внешнего источника энтропии).
2. Возвести это число X_i в квадрат, в результате чего, мы получим $2n$-разрядное число Y_i.

3. Следующее число X_{i+1} получим, составив его n-разрядное представление, выбрав средние n разрядов из числа Y_i.

Например, если начальное число $X_0 = 3485$, то $Y_1 = 3485^2 = 12145225$, $X_1 = 1452$, а $X_2 = 1083$, и т.д.

В качестве начального числа для этого алгоритма часто берут рациональное число в десятичной записи. Например, $X_0 = 0{,}3485$, $X_1 = 0{,}1452$, X_2=0,1083 и т.д.

Очевидно, что данный метод может быть реализован с помощью операций деления нацело и взятия остатка от деления предыдущего члена последовательности.

Недостаток этого метода — наличие корреляции между числами последовательности, а в ряде случаев случайность вообще может отсутствовать. Например, если $X_0 = 0{,}4500$, $X_1 = 0{,}2500, X_2 = 0{,}2500, X_3 = 0{,}2500$ и т.д. Кроме того, данный метод обладает малым периодом и сейчас представляет интерес лишь в историческом аспекте.

3.2. Линейный конгруэнтный метод

Одним из простых и популярных методов сейчас является *линейный конгруэнтный метод* (ЛКМ), предложенный Д.Г. Лехмером в 1949 году. В его основе лежит выбор четырех ключевых чисел:

- $m > 0$, модуль;
- $0 \le a \le m$, множитель;
- $0 \le c \le m$, приращение (инкремент);
- $0 \le X_0 \le m$, начальное значение.

Определение 3.1. Последовательность ПСЧ, получаемая по формуле:

$$X_{n+1} = (aX_n + c) \bmod m, n \ge 1, \qquad (3.1)$$

называется *линейной конгруэнтной последовательностью* (ЛКП). Ключом для неё служит X_0.

В данной формуле крайне важен удачный выбор параметров. Например, для $X_0 = 7, a = 8, c = 9,\ m = 10$ получим последовательность

7, 5, 9, 1, 7, 5, 9, 1,…

То есть последовательность совсем не выглядит «случайной». На данном примере проиллюстрировано, что конгруэнтная последовательность всегда зацикливается. Более того, далее будет показано, что это свойство есть у всех последовательностей вида $X_{n+1} = f(X_n)$. На рисунках 3.1-3.2 приведены графики ЛКП длиной 500 чисел. Как видно из рисунка, даже небольшое изменение параметра функции приводит к появлению короткого периода.

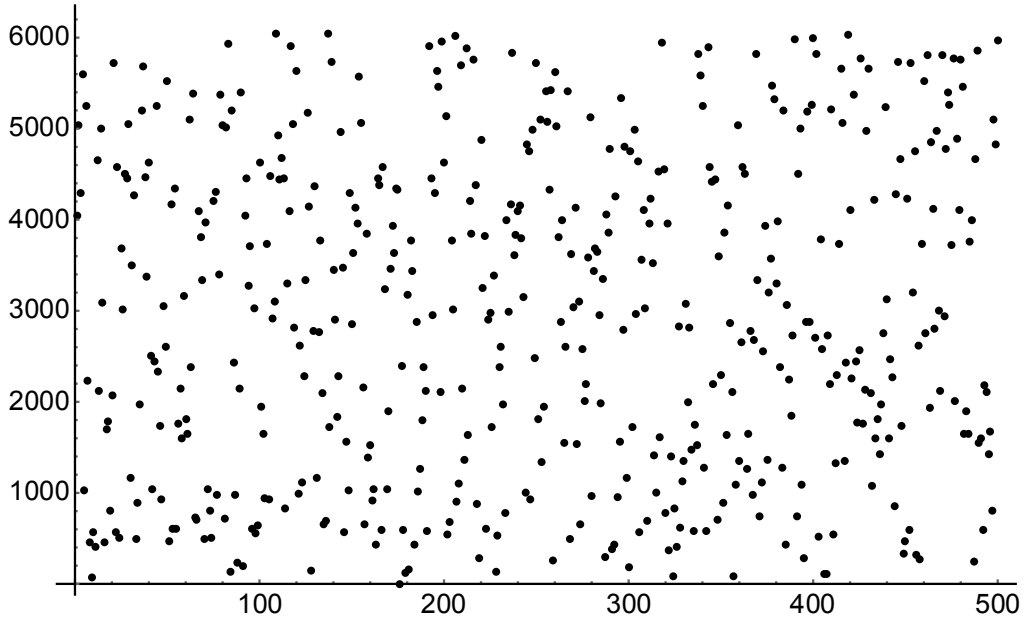

Рисунок 3.1. График ЛКП для $X_0 = 7, a = 106, c = 1283, m = 6075$.

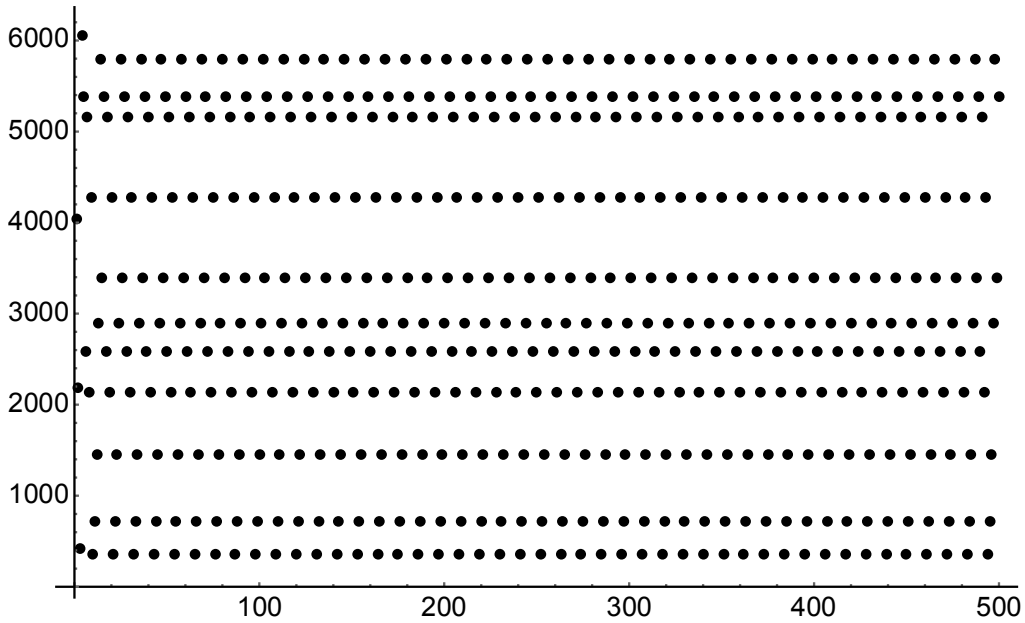

Рисунок 3.2. График ЛКП для $X_0 = 7, a = 105, c = 1283, m = 6075$.

Кроме того, у ЛКП есть еще один явный недостаток – наличие «решетчатой» структуры последовательности чисел. Этот эффект появляется в виде рас-

положения чисел на некоторых наклонных прямых, если изображать числа на плоскости. Типичный пример «решетчатой» структуры последовательности чисел (ПЧ) изображен на рисунке 3.3. Данный недостаток обусловлен свойствами операций в конечном поле, применяемых в формуле 3.1.

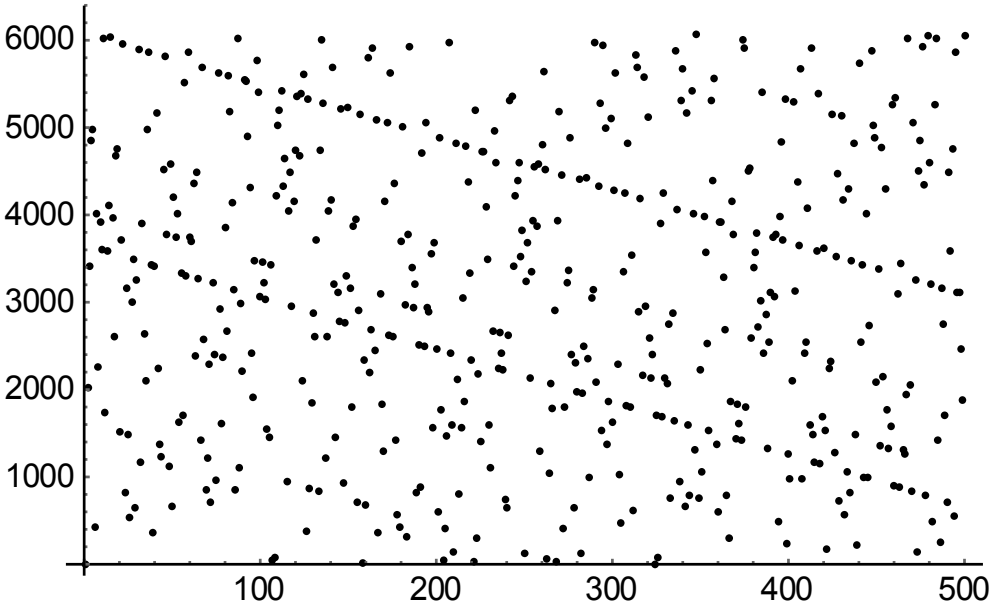

Рисунок 3.3. График ЛКП для параметров: $X_0 = 7, a = 106, c = 1284, m = 6075$.

Существует обобщение формулы ЛКМ:

$$X_{n+1} = \left(a^k X_n + \frac{(a^k-1)}{(a-1)} c\right) \bmod m, k \geq 0, n \geq 0. \qquad (3.2)$$

Рассмотрим задачу правильного выбора числа m. Во-первых, m должно быть достаточно большим. Например, для $m = 2$, последовательность в лучшем случае будет иметь вид 0,1,0,1….

Во-вторых, значение m разумно выбирать равным 2^q, где q — число битов в машинном слове, поскольку это позволяет не применять деление по модулю в формуле 3.2. Однако значение m должно удовлетворять дополнительным условиям, о которых сказано ниже.

Множитель будем искать из условий максимальности длины периода. Нужно учитывать, что требование к длине периода не единственное. Так, например, при $a = c = 1$ последовательность примет вид:

$$X_{n+1} = (X_n + 1) \bmod m.$$

Она имеет максимальный период, но не является случайной.

Теорема 3.1. Линейная конгруэнтная последовательность, определенная числами *m, a, c* и X_0, имеет период длиной *m* тогда и только тогда, когда:

1) числа *c* и *m* взаимно простые;
2) $a - 1$ кратно *p* для некоторого простого *p*, являющегося делителем *m*;
3) $a - 1$ кратно 4, если *m* кратно 4.

Доказательство приведено в [1, Том 2. п. 3.2.1.2].

Рассмотрим некоторые дополнительные условия на ЛКМ, которые не вошли в теорему 3.1, так как данная теорема определяет необходимые требования для максимальности периода, но ничего не говорит о качестве полученной ЛКП.

Определение 3.2. Потенциал линейной конгруэнтной последовательности с максимальным периодом определяется как наименьшее целое число *s*, такое, что $(a-1)^s = 0$ по модулю *m*.

Положим $X_0 = 0$. Это предположение допустимо, так как при соблюдении условий теоремы 3.1 число 0 встретится в последовательности хотя бы один раз в периоде. При этом предположении формула 3.2 сводится к

$$X_n = \frac{(a^n - 1)}{(a-1)} c \bmod m;$$

и, если разложить выражение $a^n - 1$ по биномиальной формуле, получится

$$X_n = c\left(n + \binom{n}{2}(a-1) + \ \ldots\ + \binom{n}{s}(a-1)^{s-1}\right) \bmod m. \quad (3.3)$$

В силу кратности чисел $(a-1)^s, (a-1)^{s+1}$ значению m их можно исключить из формулы.

Дальнейший анализ сводится к рассмотрению свойств разности $X_{n+1} - X_n$ для различных значений *a* и потенциала.

Например, если $a = 1$, потенциал $s = 1$, то $X_n = cn \bmod m$, и последовательность совсем не выглядит случайной. Если потенциал $s = 2$, то $X_n = cn + c(a-1)\binom{n}{2}$, и снова последовательность не выглядит случайной. Действительно, в этом случае $X_{n+1} - X_n = c + c(a-1)n$, что является простой линейной зависимостью от *n* между последовательно сгенерированными числами и дает явную «решетчатую» структуру ЛКП.

Если потенциал равен 3, то последовательность становится более или менее похожей на случайную, но все еще существует высокая степень зависимости X_n, X_{n+1}, X_{n+2}. Тесты показывают, что последовательности с потенциалом 3 также не лишены проблемы «решётчатости». Сообщалось, что приемлемые результаты были получены в некоторых случаях при потенциале, равном 4, но это оспаривалось другими исследователями. Можно предположить, что последовательности с потенциалом 5 и выше обладают достаточно хорошими случайными свойствами.

Замечание. Отметим, что высокий потенциал является необходимым, но не достаточным условием случайности. Понятие потенциала используется для исключения несостоятельных генераторов, но не для безусловного принятия генераторов с высоким потенциалом. Далее мы познакомимся с другими тестами для признания ПСЧ близкими к случайным.

Ниже приведена таблица с некоторыми константами для линейных конгруэнтных генераторов, взятая из [8].

Таблица 3.1. Параметры ЛКГ для формулы 3.1.

(a, c, m)	(a, c, m)	(a, c, m)
(106,1283,6075)	(625,6571,31104)	(1277,24749,117128)
(211,1663,7875)	(1541,2957,14000)	(2041,25673,121500)
(421,1663,7875)	(1741,2731,12960)	(2311,25367,120050)
(430,2531,11979)	(1291,4621,21870)	(1597,51749,244944)
(936,1399,6655)	(205,29573,139968)	(2661,36979,175000)
(1366,1283,6075)	(421, 17117,81000)	(4081,25673,121500)
(171,11213,53125)	(1255,6173,29282)	(3661,30809,145800)
(859,2531,11979)	(281,28411,134456)	(3613,45289,214326)
(419,6173,29282)	(1093,18257,86436)	(1366,150889,714025)
(967,3041,14406)	(421,54773,259200)	(8121,28411,134456)
(141,28411,134456)	(1021,24631,116640)	(4561,51349,243000)

При реализации ЛКМ на компьютере необходимо помнить о возможности переполнения стандартного машинного слова, используемого для хранения значения вычислений по формулам 3.1,3.2.

Линейный конгруэнтный метод можно обобщить до *полиномиального конгруэнтного метода* (ПКМ). Например, существуют следующие разновидности ПКМ.

Квадратичный конгруэнтный генератор:

$$X_n = (aX_{n-1}^2 + bX_{n-1} + c) \bmod m.$$

Кубический конгруэнтный генератор:

$$X_n = (aX_{n-1}^3 + bX_{n-1}^2 + cX_{n-1} + d) \bmod m.$$

Полиномиальный генератор степени *r*:

$$X_n = \sum_{i=0}^{r} a_i X_{n-1}^i \bmod m, \quad r \geq 1.$$

Преимуществом линейных конгруэнтных генераторов является простота их реализации и быстрота. Однако их нельзя использовать в криптографии, так как их легко «взломать». То есть, имея последовательность чисел, полученную таким генератором, оппонент может восстановить параметры генератора, затратив минимум усилий. Впервые это было показано Джимом Ридсом [42], а затем Джоан Бояр [43].

Тем не менее, ЛКМ являются полезными для не криптографических приложений. Например, в моделировании или в игровых приложениях.

3.3. Аддитивный ГПСЧ

Идею рекурсивного вычисления значения можно обобщить до формулы, использующей два предыдущих значения последовательности. Например, мы можем рекуррентно вычислять значение X_{n+1} как линейную комбинацию значения X_n и X_{n-1}. Тогда, максимальная длина последовательности в лучшем случае будет равна m^2, так как последовательность не будет повторяться, пока не будет получено равенство $(X_{n+l}, X_{n+l+1}) = (X_n, X_{n+1})$. Простейшая последовательность, в которой X_{n+1} зависит более чем от одного предыдущего значений, это последовательность Фибоначчи

$$X_{n+1} = (X_n + X_{n-1}) \bmod m.$$

Данный генератор рассматривался в начале 1950-х, и обычно он дает длину периода, большую, чем m. Однако числа, получаемые с помощью рекуррентного соотношения Фибоначчи, недостаточно случайны. Но можно сделать еще один шаг обобщения и использовать формулу

$$X_{n+1} = \left(X_{n-k} + X_{n-j}\right) \bmod m,\ j > k \geq 1.$$

Например, Дж. Ж. Митчел и Д.Ф. Мур в 1958 году предложили последовательность, определенную так:

$$X_n = (X_{n-24} + X_{n-55}) \bmod m, n \geq 55, \qquad (3.4)$$

где m – четное число, а $X_0, \ldots, X_{54}$ – произвольные целые числа.

Числа k и j обычно называют запаздыванием, а последовательность 3.4, – последовательностью Фибоначчи с запаздыванием. Для ПСЧ Фибоначчи с запаздыванием справедлива

Теорема 3.2. Если многочлен $x^k + x^j + 1$ является примитивным многочленом над полем $GF(2)$, то последовательность, формируемая аддитивным ГПСЧ Фибоначчи с запаздыванием, имеет максимальный период, равный $2^{\log_2 m - 1}(2^{\max\{k,j\}} - 1)$.

В частности, для формулы 3.4 справедлива оценка длины периода $2^{31}(2^{55} - 1)$, при $m = 2^{32}$.

Приведем другие варианты запаздываний (k, j) для аддитивного ГПСЧ [1, 53]: (9,49), (19,58), (18,65), (25,73), (38,89), (2,93), (21,94), (11,95), (37, 100), (33,118), (10,111), (37,124), (29,132), (52,145), (57,134), (83, 258) (107,378), (273, 607), (1029, 2281), (576, 3217), (4187, 9689), (7083, 19937), (9739, 23209). При больших запаздываниях данный метод становится неэффективным из-за высоких требований к памяти.

Были также предложены модификации такого генератора, в которых вместо сложения использовалось умножение. Эти генераторы хорошо себя показали и с 1958 года применялись в различных прикладных задачах. В частности, один из вариантов такого ГПСЧ используется в пакете Matlab. Однако в 1990-е годы они провалились на спектральном тесте.

Замечание. В хорошем источнике случайных чисел неравенства $X_{n-1} < X_n < X_{n+1}$ будут встречаться примерно один раз из шести, так как каждое из шести возможных отношений порядка X_{n-1}, X_n, X_{n+1} должно иметь одну и ту же веро-

ятность. Можно показать, что приведенный порядок никогда не возникнет, если использовать последовательность Фибоначчи $X_{n+1} = (X_n + X_{n-1}) \bmod m$.

Данный аддитивный генератор имеет обобщение: можно вычислять очередное значение последовательности как линейную комбинацию предыдущих k элементов этой последовательности. Когда $m = p$ – простое число, то согласно теории конечных полей можно найти множители $a_1, \dots, a_k$, такие, что последовательность, определенная формулой

$$X_n = (a_1 X_{n-1} + \cdots + a_{k-1} X_{n-k+1} + a_k) \bmod p, \qquad (3.5)$$

будет иметь максимально возможный период равный $p^k - 1$. То есть справедлива

Теорема 3.3. Если константы $a_1, a_2, \dots, a_k$ таковы, что многочлен $x^k - a_1 x^{k-1} - \cdots - a_k$ является примитивным над полем $GF(p)$ и хотя бы один из элементов $X_0, X_1, \dots, X_k$ не равен нулю, то период генератора равен $p^k - 1$.

Однако выбор «хороших» коэффициентов a_i – нетривиальная задача, требующая от исследователя аналитических вычислений. Дело в том, что константы $a_1, \dots, a_k$ обладают подходящими свойствами тогда и только тогда, когда полином

$$\Phi(x) = x^k - a_1 x^{k-1} - \cdots - a_k$$

является первообразным полиномом по модулю p, что выполняется тогда и только тогда, когда корень этого полинома есть первообразный элемент поля с p^k элементами. Проверка этого может столкнуться с необходимостью разложения на простые множители большого числа, поэтому, как правило, для проверки «случайности» ГПСЧ, основанного на формуле 3.5, используют критерии из стандартного набора проверки качества ГПСЧ.

На основе аддитивных ГПСЧ создано несколько алгоритмов шифрования. В частности, далее в третьем разделе данной работы будут приведены алгоритмы **Fish**, **Pike** и **Mush,** основанные на аддитивном генераторе с запаздыванием.

3.4. Инверсный конгруэнтный генератор

Еще одной разновидностью конгруэнтных генераторов является алгоритм, вычисляющий обратную функцию от линейной комбинации предыдущих членов последовательности. В простом случае, схема создания нового числа явля-

ется обратной конгруэнтной последовательностью, предложенной Эйченауэром и Лехном в 1986 году:

$$X_{n+1} = (aX_n^{-1} + c) \bmod p, \text{если } X_n \neq 0, \quad (3.6а)$$

$$X_{n+1} = c, \quad \text{если } X_n = 0 \quad (3.6б)$$

Данный генератор получил название *инверсный конгруэнтный генератор*. Естественно, наличие обратного элемента возможно только в поле $GF(p)$, где p – простое число. X_n принимает значения из множества $\{0,1,\dots,p-1,\infty\}$, а обращение 0 определено как $0^{-1} = \infty$. В других случаях $X^{-1}X = 1 \bmod p$. Параметры подбираются таким образом, чтобы $\text{НОД}(X_0, p) = 1, \text{НОД}(a, p) = 1$.

Теорема 3.4. Максимальный период p последовательность, генерируемая инверсным конгруэнтным генератором, определяемым формулами 3.6а-3.6б, имеет тогда и только тогда, когда многочлен $\Phi(x) = x^2 - cx - a$ является примитивным многочленом поля $GF(p)$.

Доказательство теоремы приведено в [54].

Данный генератор показывает хорошие показатели равномерности и может быть обобщен до полиномиальной формулы

$$X_{n+1} = \sum_{i=0}^{r} a_i X_n^{i-r} \bmod p.$$

Например, формула

$$X_{n+1} = aX_n^{-3} + c \bmod p$$

показывает хорошие характеристики ПСЧ даже при «слабых» значениях a и c или при изменении исходных параметров (см. рисунок 3.4). Кроме того, ПСЧ, полученные данным методом, не имеют нежелательных статистических отклонений [47, 48] и успешно проходят большинство тестов на случайность.

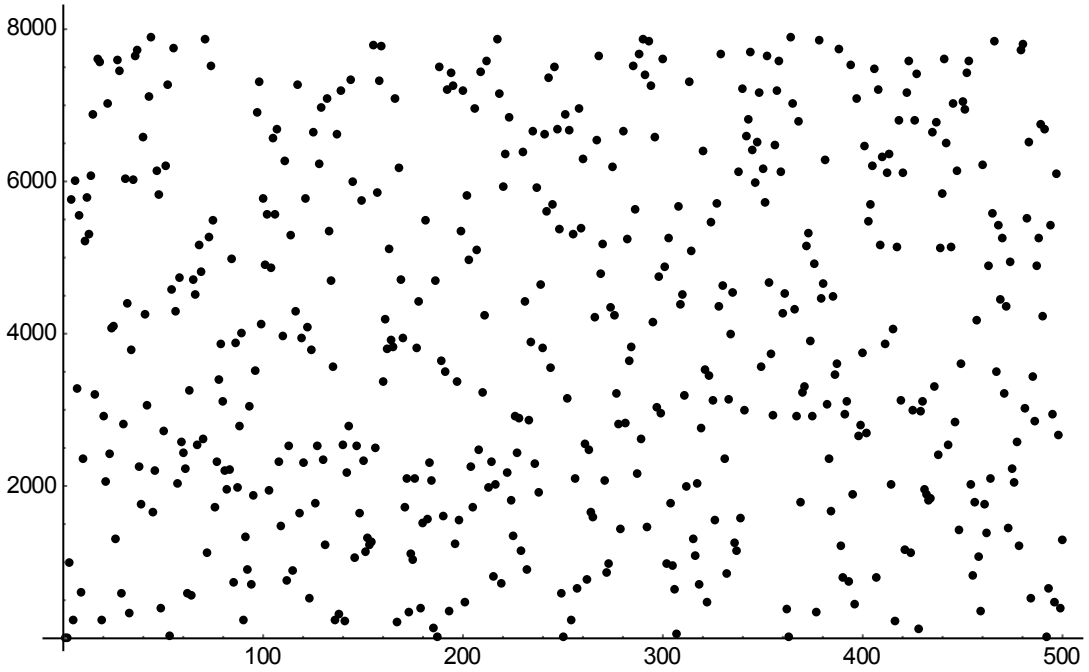

Рисунок 3.4. График ПСЧ $X_{n+1} = aX_n^{-3} + c \bmod p$ с параметрами $X_0 = a = c = 1, p = 104729$.

Основной недостаток инверсных генераторов – это трудоемкость операции обращения элемента конечного поля $GF(p)$. В среднем она зависит от значения p как $O(N^2)$, где $N = \log_2 p$ [50].

3.5. Регистр сдвига с обратной линейной связью. Метод М-последовательности.

Следующий класс ГПСЧ основан на идее преобразования двоичного представления некоторого числа. Такие генераторы имеют некоторые преимущества, как, например, скорость генерации таких чисел, хорошие статистические свойства ПСЧ, а также возможность простой реализации на аппаратном уровне.

Определение 3.3. *Регистр сдвига* – упорядоченный набор битов, допускающий операцию изменения позиций битов на одну и ту же величину влево или вправо.

Определение 3.4. *Регистр сдвига с обратной линейной связью* (РСЛОС) – регистр сдвига битовых слов, у которого входной (вдвигаемый) бит является линейной функцией остальных битов. Вдвигаемый вычисленный бит заносится в ячейку с номером 0. Количество ячеек p называют длиной регистра.

Для натурального числа p и $a_1, a_2, \dots, a_{p-1}$, принимающих значения 0 или 1, определяют рекуррентную формулу[1]

$$X_{n+p} = a_{p-1}X_{n+p-1} + a_{p-2}X_{n+p-2} + \cdots + a_1X_{n+1} + X_n, \qquad (3.7)$$

Как видно из формулы, для РСЛОС функция обратной связи является линейной булевой функцией от состояний всех или некоторых битов регистра.

Одна итерация алгоритма, генерирующего последовательность, состоит из следующих шагов:

1. Содержимое ячейки $p-1$ формирует очередной бит ПСП битов.
2. Содержимое ячейки 0 определяется значением функции обратной связи, являющейся линейной булевой функцией с коэффициентами $a_1, a_2, \dots, a_{p-1}$. Его вычисляют по формуле 3.7.
3. Содержимое каждого *i*-го бита перемещается в $(i+1)$-й, $0 \le i < p-1$.
4. В ячейку 0 записывается новое содержимое, вычисленное на шаге 2.

Наименьшее положительное целое *N*, такое, что $X_{n+N} = X_n$ для всех значений *n* называют *периодом последовательности*. Эту последовательность называют М-последовательностью, если её период равен $(2^p - 1)$. Буква М в обозначении М-последовательности является первой буквой английского слова “maximum”. Период любой последовательности, сгенерированной по приведенной рекуррентной формуле, не может быть больше $(2^p - 1)$. Поэтому, если ряд с периодом $(2^p - 1)$ существует, это ряд с наибольшим периодом.

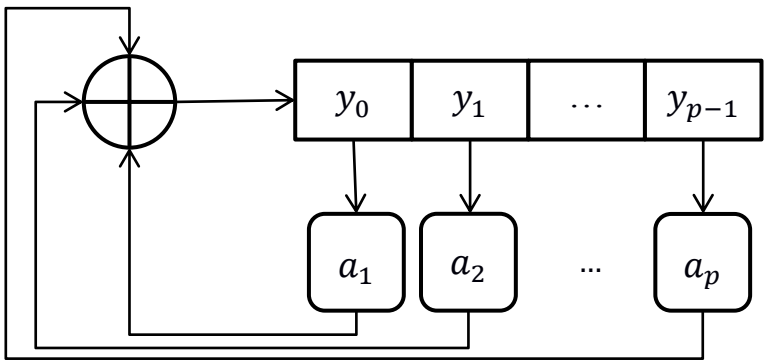

Рисунок 3.5. Схема РСЛОС для многочлена $\Phi(x) = x^p + a_{p-1}x^{p-1} + a_{p-2}x^{p-2} + a_1x + 1$.

[1] Здесь и далее рассматриваются только операции в конечном поле $GF(2)$, поэтому знаком + обозначена операция суммирования по модулю 2.

При анализе РСЛОС используется математический аппарат теории конечных полей. Свойства выдаваемой РСЛОС последовательности тесно связаны со свойствами многочлена

$$\Phi(x) = a_p x^p + a_{p-1} x^{p-1} + \cdots + a_1 x + 1 \qquad (3.8)$$

над полем $GF(2)$. Его ненулевые коэффициенты называются *отводами*, как и соответствующие ячейки регистра, поставляющие значения аргументов функции обратной связи. Такой многочлен называется *образующим многочленом* РСЛОС.

Общий вид формулы следующего состояния регистра в момент времени $t+1$, соответствующего образующему многочлену $\Phi(x)$ степени p:

$$Y(t+1) = T^k Y(t), \qquad (3.9)$$

где $Y(t)$ – вектор состояния регистра в момент времени t; T – квадратная матрица порядка p вида

$$\begin{pmatrix} a_1 & a_2 & \dots & a_{p-1} & a_p \\ 1 & 0 & \dots & 0 & 0 \\ 0 & 1 & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 1 & 0 \end{pmatrix}.$$

В частности, при $k = 1$, получаем схему генерации ПСП по формуле (3.7), в которой $Y(t) \coloneqq (X_n, X_{n+1}, \dots, X_{n+p-1})$.

Пример. Многочлену $\Phi(x) = x^{12} + x^6 + x^4 + x + 1$ соответствует формула вычисления первой ячейки регистра

$$y_0(t+1) = y_0(t) + y_1(t) + y_4(t) + y_6(t) + y_{12}(t).$$

Остальные значения ячеек получаются по формуле $y_{i+1}(t+1) = y_i(t), i = 0,2 \dots, p-2$.

Замечание. При использовании данного метода в качестве образующего многочлена применяют один из многочленов, приведенных в справочной литературе. Некоторые из таких многочленов, взятые из [10], представлены в таблице 3.2. Запись $(n_1, n_2, \dots, n_p)$ в этой таблице следует трактовать как многочлен

$$x^{n_1} + x^{n_2} + \cdots + x^{n_p}.$$

Таблица 3.2. Коэффициенты примитивных многочленов по модулю 2.

(1,0)	(12,6,4,1,0)	(21,2,0)	(31,7,0)	(37,5,4,3,2,1,0)
(2,1,0)	(13,4,3,1,0)	(22,1,0)	(31,13,0)	(38,6,5,1,0)
(3,1,0)	(14,5,3,1,0)	(23,5,0)	(32,7,6,2,0)	(39,4,0)
(4,1,0)	(15,1,0)	(24,4,3,1,0)	(32,7,5,3,2,1,0)	(40,5,4,3,0)
(5,2,0)	(16,5,3,2,0)	(25,3,0)	(33,13,0)	(41,3,0)
(6,1,0)	(17,3,0)	(26,6,2,1,0)	(33,16,4,1,0)	(42,7,4,3,0)
(7,1,0)	(17,5,0)	(27,5,2,1,0)	(34,8,4,3,0)	(42,5,4,3,2,1,0)
(7,3,0)	(17,6,0)	(28,3,0)	(34,7,6,5,2,1,0)	(43,6,4,3,0)
(8,4,3,2,0)	(18,7,0)	(29,2,0)	(35,2,0)	(44,6,5,2,0)
(9,4,0)	(18,5,2,1,0)	(30,6,4,1,0)	(36,11,0)	(45,4,3,1,0)
(10,3,0)	(19,5,2,1,0)	(31,3,0)	(36,6,5,4,2,1,0)	(46,8,7,6,0)
(11,2,0)	(20,3,0)	(31,5,0)	(37,6,4,1,0)	(46,8,5,3,2,1,0)

3.5.1. Свойства М-последовательности (РСЛОС)

Напомним определение примитивного многочлена.

Определение 3.5. Многочлен $\Phi(x)$ степени p называется *примитивным*, если он не делит нацело ни один многочлен вида $x^s - 1$, где $s < 2^p - 1$. Наименьшее натуральное число *e,* при котором $x^e - 1$ делится на $\Phi(x)$ без остатка называется *показателем* многочлена $\Phi(x)$.

Свойства последовательности битов, генерируемой РСЛОС, зависят от образующего многочлена:

- Если старший коэффициент ассоциированного многочлена $a_p = 0$, то периодичность генерируемой последовательности может проявляться не сразу.
- Если $a_p = 1$, то соответствующая последовательность называется неособой. Она будет периодичной с самого начала, т.е. равенство $X_{N+i} = X_i$ выполнено для всех i, а не только для достаточно больших.

Особо выделим следующее

Свойство 3.1. Для РСЛОС $Y(t+1) = T^k Y(t)$, с образующим неприводимым многочленом, формируемая последовательность имеет максимальный период $N = 2^p - 1$, тогда и только тогда, когда числа N и k взаимно просты.

Примечание. Два целых числа являются взаимно простыми, если у них нет общих делителей, кроме единицы.

Следствием последнего свойства является

Теорема 3.5. Если $k = 1$, то примитивность образующего многочлена является необходимым и достаточным условием того, что ПСП, генерируемая РСЛОС будет иметь максимальный период $N = 2^p - 1$.

3.5.2. Пятипараметрический метод

Данный метод является частным случаем РСЛОС, использует характеристический многочлен из 5 членов и позволяет генерировать последовательности w-битовых двоичных целых чисел в соответствии со следующей рекуррентной формулой: $X_{n+p} = X_{n+q_1} + X_{n+q_2} + X_{n+q_3} + X_n,\ n = 1,2,3, \ldots$.

Параметры (p, q_1, q_2, q_3, w) и $X_1, \ldots, X_p$, первоначально задают как начальный вектор. Примеры параметров p, q_1, q_2, q_3 с наибольшим периодом приведены в таблице 3.3.

Таблица 3.3. Параметры пятипараметрического метода.

p	q_1	q_2	q_3
89	20	40	69
107	31	57	82
127	22	63	83
521	86	197	447
607	167	307	461
1279	339	630	988
2203	585	1197	1656
2281	577	1109	1709
3217	809	1621	2381
4253	1093	2254	3297

3.5.3. Комбинированный метод Таусворта

При генерации чисел методом Таусворта [6] используют рекуррентную формулу

$$x_{n+p} = x_{n+q} + x_n, \qquad n = 0,1,2, \ldots,$$

где x_0, x_1, x_2, … - соответствующая М-последовательность.

При использовании такой М-последовательности последовательность w-битовых целых чисел, называемую простой последовательностью Таусворта с параметрами (p, q, t), получают по формуле

$$X_n = x_{nt}x_{nt+1} \ldots x_{nt+w-1}, \qquad n = 0,1,2, \ldots,$$

где t – натуральное число взаимно простое с периодом $(2^p - 1)$ М-последовательности; w – длина слова, не превышающая p бит.

Период этой последовательности составляет $(2^p - 1)$.

Упражнение. Получить первые четыре числа последовательности комбинированным методом Таусворта для параметров $(p, q, t) = (4,1,4)$, с длиной числа – 4 бит.

Если имеется J последовательностей Таусворта с одной и той же длиной слова w, комбинированный метод Таусворта генерирует последовательность псевдослучайных чисел $\{X_n\}$ как результат побитовой операции сложения по модулю 2 («исключающее ИЛИ», которое мы будем обозначать символом «⊕») соответствующих двоичных векторов последних членов этих последовательностей:

$$X_n = X_n^1 \oplus X_n^2 \oplus X_n^3 \oplus \ldots \oplus X_n^J, n = 0,1,2, \ldots$$

Параметры и начальный вектор комбинированной последовательности Таусворта представляют собой комбинацию параметров и начального вектора каждой простой последовательности Таусворта. Если периоды J простых последовательностей Таусворта являются взаимно простыми, то период комбинированной последовательности Таусворта равен произведению периодов J последовательностей.

3.6. Регистр сдвига с обратной связью по переносу

Регистр сдвига с обратной связью по переносу (РСОСП) в отличие от РСЛОС имеет дополнительный регистр – *регистр переноса*. Данный регистр

является не битом, а числом, которое есть результат преобразования битов, полученных из основного регистра. Это преобразование называется функцией обратной связи.

Схема работы алгоритма следующая. На каждой итерации биты отводной последовательности складываются друг с другом и содержимым регистра переноса. Остаток от деления по модулю 2 становится новым нулевым битом основного регистра сдвига. А целая часть результата, делённого на два, становится новым значением регистра переноса.

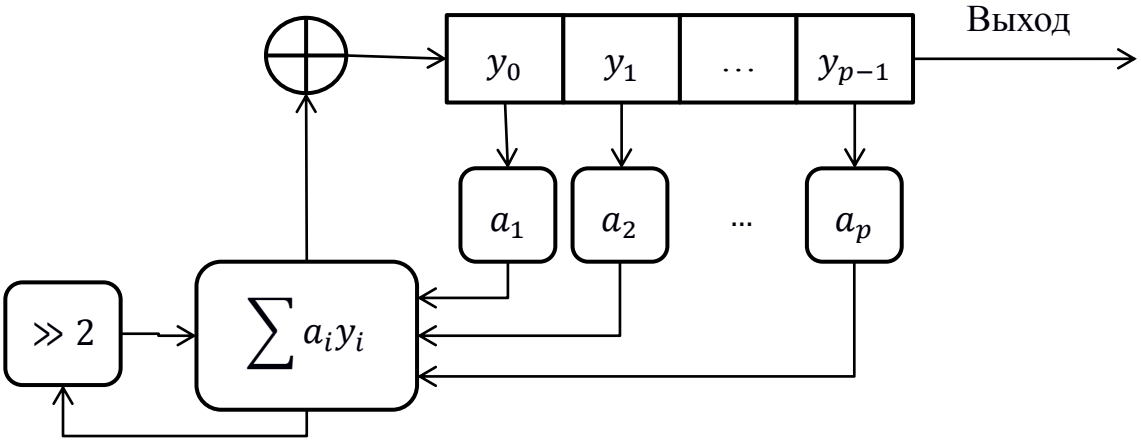

Рисунок 3.6. Схема РСОСП.

Таким образом, работа РСОСП описывается следующими формулами:

$$\sigma = a_1 y_0(t) + \cdots + a_p y_{p-1}(t) + m(t),$$

$$y_{i+1}(t+1) = y_i(t), \qquad i = 0, \ldots, p,$$
$$y_0(t+1) = \sigma \bmod 2,$$
$$m(t+1) = \lfloor \sigma(t)/2 \rfloor.$$

Примечание. Регистр переноса является числом, в двоичном представлении занимающим не менее $\log_2 t$ битов, где t – число ответвлений. Также отметим, что остаток от деления на два равен последнему биту числа, а целая часть от деления на два равна содержимому регистра переноса, представленному в двоичном виде, сдвинутому на одну позицию вправо.

Максимальный период последовательности этого генератора равен $q - 1$, где q определяется формулой

$$q = 2 \cdot a_1 + 2^2 \cdot a_2 + 2^3 \cdot a_3 + \cdots + 2^n \cdot a_n - 1.$$

В отличие от большинства РСЛОС, для РСОСП существует задержка, прежде чем он перейдёт в циклический режим, то есть начнёт генерировать циклически повторяемую последовательность. В зависимости от выбранного начального состояния возможны 4 различных случая:

1. Начальное состояние может оказаться частью максимального периода.
2. Начальное состояние может перейти в последовательность максимального периода, после некоторой начальной задержки.
3. Начальное состояние может после начальной задержки породить последовательность нулей.
4. Начальное состояние может после начальной задержки породить последовательность единиц.

3.7. Нелинейная комбинация РСЛОС. Генератор Геффа.

Определение 3.6. *Алгебраическая нормальная форма функции* $f(x_1, x_2, \dots, x_n)$ – запись функции суммой по модулю 2 произведений порядков m независимых переменных $0 \le m \le n$. *Нелинейным порядком* функции f называется максимальный порядок членов в записи её алгебраической нормальной формы.

Пример. Функция $f(x_1, x_2, x_3, x_4) = x_1 \oplus x_2 \oplus x_2 x_3 x_4$ имеет нелинейный порядок 3.

Предположим теперь, что у нас n регистров сдвига с линейной обратной связью $f_1, f_2, \dots, f_n$, их длины $L_1, L_2, \dots, L_n$ попарно различны и больше двух. Подобно комбинированному методу Таусворта, мы можем объединить эти РСЛОС, но уже при помощи нелинейной функции *f:*

$$f(x_1, x_2, \dots, x_n) = f(f_1, f_2, \dots, f_n).$$

Тогда *линейная сложность* (понятие линейной сложности описано в п. 4.1) потока ключей равна $f(L_1, L_2, \dots, L_n)$. Если $L_1, L_2, \dots, L_n$ – попарно взаимно простые числа, то длина периода последовательности равна:

$$\left(2^{L_1} - 1\right) \cdot \left(2^{L_2} - 1\right) \dots \left(2^{L_n} - 1\right).$$

Генератор Геффа является примером нелинейной комбинации РСЛОС. В этом генераторе используются три РСЛОС, объединённые нелинейным образом. Длины этих регистров L_1, L_2, L_3 – попарно простые числа. Нелинейная функция генератора:

$$f(x_1, x_2, x_3) = x_1 x_2 \oplus (1 + x_2) x_3 = x_1 x_2 \oplus x_2 x_3 \oplus x_3.$$

Длина периода ПСЧ: $\left(2^{L_1} - 1\right) \cdot \left(2^{L_2} - 1\right) \cdot \left(2^{L_3} - 1\right)$.

Линейная сложность: $L = L_1 \cdot L_2 + L_2 \cdot L_3 + L_3$.

3.8. Генератор «стоп-пошёл»

Генератор «*стоп-пошёл*» является нелинейным объединением двух регистров сдвига – РСЛОС-1 и РСЛОС-2. При этом РСЛОС-2 работает только если в момент времени $t - 1$ выходное значение РСЛОС-1 было равно 1. Запуск-останов генератора РСЛОС-2 называется его *тактированием*. Таким образом, РСЛОС-1 *тактирует* генератор РСЛОС-2.

Данный ГПСЧ имеет большую длину периода, нежели РСЛОС-1 или РСЛОС-2, используемые по отдельности. Однако недостатком такого генератора являются задержки при генерации чисел, зависящие от длины последовательности нулей, выдаваемых РСЛОС-1.

В целях увеличения криптостойкости, а также для избавления от недостатка «задержек», был предложен чередующийся генератор «стоп-пошёл», состоящий из трех РСЛОС. Здесь РСЛОС-1 управляет тактовой частотой 2-го и 3-го регистров, то есть РСЛОС-2 меняет своё состояние, когда выход РСЛОС-1 равен единице, а РСЛОС-3 – когда выход РСЛОС-1 равен нулю. Выходом генератора является операция побитового сложения выходов РСЛОС-2 и РСЛОС-3. Линейная сложность и длина периода данного генератора, по уверениям авторов [55], достаточно большая.

3.9. Пороговый генератор

Этот генератор является одним из обобщений генератора Геффа. *Пороговый генератор* использует результат работы любого числа РСЛОС. Если имеется n РСЛОС со взаимно простыми длинами периодов и примитивными многочленами обратной связи, то выход порогового генератора получается в результате голосования: если более половины генераторов РСЛОС имеют на выходе 1, то выход порогового генератора равен единице, и равен нулю в другом случае.

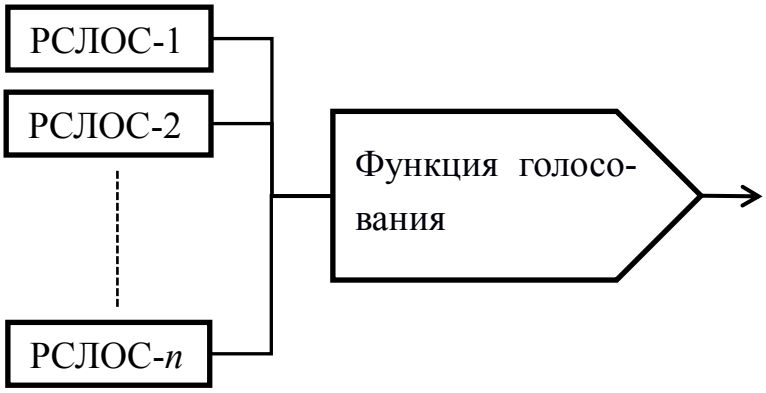

Рисунок 3.7. Схема порогового ГПСЧ.

Например, для трех РСЛОС выход генератора можно вычислять по формуле:

$$b(t) = y_p^1(t)y_p^2(t) + y_p^1(t)y_p^3(t) + y_p^2(t)y_p^3(t).$$

3.10. Многоскоростной генератор с внутренним произведением

Многоскоростной генератор с внутренним произведением [10] состоит из двух РСЛОС с разными тактовыми частотами: РСЛОС-1 выдает один бит за d тактов времени, а РСЛОС-2 выдает d бит. Отдельные биты каждого из РСЛОС объединяются операцией двоичного умножения, а затем, для получения выходного бита генератора, складываются по модулю 2.

3.11. Каскад Голлмана

Каскад Д. Голлмана [56] является обобщением генератора «стоп-пошел», однако состоит не из двух, а из последовательности РСЛОС, каждый из которых тактируется предыдущим РСЛОС (кроме начального). Если выходом РСЛОС-1 в момент времени t является 1, то тактируется РСЛОС-2. Если выходом РСЛОС-2 в момент времени t является 1, то тактируется РСЛОС-3, и так далее. Выход последнего РСЛОС и является выходом генератора. При условии примитивности характеристических многочленов всех РСЛОС степени n, линейная сложность системы из k РСЛОС равна $q(2^q - 1)^{k-1}, q = 2^n - 1$.

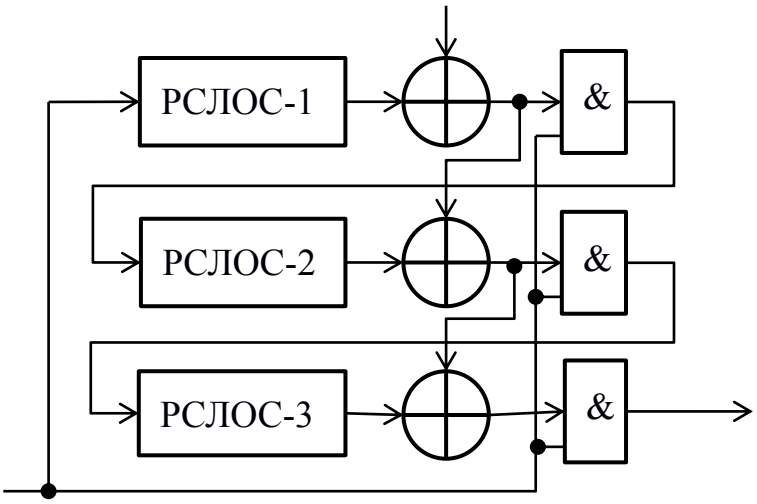

Рисунок 3.8. Схема генератора Голлмана.

Такие генераторы просты в реализации, имеют большие периоды последовательности, обладают большими линейными сложностями и хорошими статистическими свойствами. Такие генераторы чувствительны к вскрытию, называемому «запиранием» (lock-in) [11], когда криптоаналитик восстанавливает внутренние параметры генераторов по выходу последовательно, один за другим, начиная с последнего в каскаде.

Анализ данного типа генераторов показал, что с ростом количества генераторов последовательность приближается к случайной. В книге Б. Шнайера [10] приведена рекомендация использовать не менее 15 РСЛОС в каскаде.

3.12. Прореживаемый генератор

Прореживаемый генератор состоит из двух РСЛОС. При старте тактовый импульс подается на оба регистра. Если выходом РСЛОС-1 является 1, то выходом генератора является выход РСЛОС-2. Если выход РСЛОС-1 равен 0, то оба бита пропускаются и процедура повторяется заново.

Данный генератор имеет ту же проблему, что и генератор «стоп–пошёл» - задержки при выдаче последовательности битов. Например, если РСЛОС-1 выдает длинную последовательность нулей, то на выходе генератора нет результата.

Этот генератор имеет модификацию, называемую «*Самопрореживаемый генератор*». В ней вместо младших битов двух различных РСЛОС используется два младших бита одного РСЛОС. Нужно тактировать РСЛОС дважды: если первым битом будет 1, то второй бит будет выходом генератора, в противном

случае, нужно сбросить оба бита и вычислить значение РСЛОС снова. Такой генератор требует меньше памяти, но работает медленнее и имеет меньшую линейную сложность.

3.13. Алгоритм A5

Первый вариант алгоритма A5/1 был разработан в 1987 и в дальнейшем усовершенствован до A5/2, A5/3, главным образом, за счет увеличения длины ключа. A5 является потоковым шифром для шифрования данных в сети мобильной связи стандарта GSM (Group Special Mobile). Данный стандарт входит в группу стандартов безопасности цифровой сотовой связи (A2, A5/1, A5/2, A8).

Рассмотрим версию алгоритма A5/1, как ГПСЧ. Он состоит из трех РСЛОС – R1, R2, R3 размером 19, 22 и 23 бит. Выходом является сумма по модулю два всех трех РСЛОС. В A5 используется изменяемое управление тактированием: каждый из РСЛОС тактируется в зависимости от своего *бита синхронизации*, затем выполняется сложение по модулю 2 с обратной пороговой функцией битов синхронизации всех трех регистров.

Структура алгоритма A5 выглядит следующим образом.

Многочлены обратных связей:

- R1: $X^{19} + X^{18} + X^{17} + X^{14} + 1$,
- R2: $X^{22} + X^{21} + 1$,
- R3: $X^{23} + X^{22} + X^{21} + X^{8} + 1$.

Управление тактированием осуществляется следующим образом:

В каждом регистре есть биты синхронизации: R1[8], R2[10], R3[10].

1. Вычисляется функция $F(x, y, z) = xy \vee xz \vee yz$, где $x = R1[8]$ – 8-й бит R1, $y = R2[10]$ – 10-й бит R2, а $z = R3[10]$ — 10-й бит R3.
2. Сдвигаются только те регистры, у которых бит синхронизации равен F, то есть, сдвигаются регистры, бит синхронизации которых, принадлежит большинству.
3. Выходной бит системы — результат операции сложения по модулю 2 над выходными битами регистров: $R1[18] + R2[21] + R[22]$.

В целом алгоритм считается эффективным: он удовлетворяет всем известным статистическим тестам. Однако данный ГПСЧ не может быть исполь-

зован в качестве алгоритма надежного потокового шифрования из-за криптографической слабости линейных регистров сдвига. Успешные атаки на данный шифр описаны в 1999 году сначала Вагнером Голдбергом, затем группой израильских ученых Ади Шамиром, Алексом Бирюковым [44] и Дэвидом Вагнером. Ими показаны достаточно эффективные методы атаки на данный алгоритм.

3.14. Hughes XPD/KPD

Этот алгоритм был создан компанией Hughes Aircraft Corp в 1986 году для шифрования данных в армейских тактических рациях и других армейских устройствах связи [12]. Алгоритм был назван XPD (Exportable Protection Device) – Экспортируемое устройство защиты, но позднее был переименован в KPD – Устройство кинетической защиты.

Алгоритм использует 61-битовый РСЛОС. Принцип работы алгоритма следующий: 1024 различных примитивных многочлена обратной связи, а также начальное состояние РСЛОС хранятся в ПЗУ устройства шифрования. Ключ определяет один из этих многочленов и начальное состояние. В алгоритме восемь различных нелинейных фильтров, каждый из которых использует шесть отводов РСЛОС. Выходные биты этих РСЛОС образуют байт, используемый для шифрования.

3.15. Алгоритм Fish

Следующий алгоритм является композицией аддитивного ГПСЧ и идеи прореживания последовательности чисел. Название алгоритма – сокращение *Fibonacci shrinking generator* – прореживаемый генератор Фибоначчи.

Алгоритм состоит из следующих шагов:

1. Задается начальное состояние двух аддитивных генераторов A_0, B_0.
2. На i-м шаге вычисляются i-е значения по формулам
$$A_i = (A_{i-55} + A_{i-24})\ mod\ 2^{32},$$
$$B_i = (B_{i-52} + B_{i-19})\ mod\ 2^{32}.$$
3. К этим последовательностям применяется процедура прореживания в зависимости от младшего значащего бита B_i: если значение равно 1, то пара используется, если 0 – игнорируется.
4. Результатом работы генератора на j-м шаге является 32-битное слово, вычисленное по формуле:

a. $E_{2j} = C_{2j} \oplus (D_{2j} \cdot D_{2j+1})$,
b. $F_{2j} = D_{2j+1} \cdot (E_j \cdot C_{2j+1})$,
c. $K_{2j} = E_{2j} \oplus F_{2j}$,
d. $K_{2j+1} = C_{2j+1} \oplus F_{2j}$,

где C_j – последовательность используемых слов A_i, а D_j – это последовательность используемых слов B_i. K_{2j} и K_{2j+1} – пара, получаемая на выходе генератора. Знак умножения в формулах – побитовая операция умножения двоичных векторов.

3.16. Алгоритм Pike

Pike [10, 13] – более быстрая модификация алгоритма Fish. Помимо основной идеи использования аддитивного ГПСЧ, Pike использует идею порогового механизма управления движением регистров, аналогичного схеме алгоритма A5. В алгоритме используется три аддитивных генератора:

$$A_i = (A_{i-55} + A_{i-24}) \bmod 2^{32},$$
$$B_i = (B_{i-57} + B_{i-7}) \bmod 2^{32},$$
$$C_i = (C_{i-58} + C_{i-19}) \bmod 2^{32}.$$

Для генерации 32-битного слова последовательности ключей на каждой итерации рассматриваются биты переноса при сложении. Если все три одинаковы, то тактируются все три генератора. Если нет, то тактируются только два совпадающих генератора. Биты переноса сохраняются для следующей итерации. Окончательным выходом является побитовая сумма по модулю 2 выходов всех трех генераторов.

3.17. Алгоритм Mush

Mush [10] также основан на аддитивном генераторе, но использует процедуру прореживания. Основные шаги алгоритма следующие:

1. Вычисление значений аддитивных генераторов
$$A_i = (A_{i-55} + A_{i-24})\ mod\ 2^{32},$$
$$B_i = (B_{i-52} + B_{i-19})\ mod\ 2^{32}.$$
2. Если бит переноса *A* установлен, тактируется *B*. Если бит переноса *B* установлен, тактируется *A*.

3. Тактируем *A* и при переполнении устанавливаем бит переноса. Тактируем *B* и при переполнении устанавливаем бит переноса.
4. Выходом является побитовая сумма по модулю 2 выходов *A* и *B*.

3.18. ГПСЧ на базе клеточного автомата

В [16, 18] описан алгоритм генерации ПСЧ на базе *клеточного автомата*. Этот ГПСЧ был предложен Стивеном Вольфрамом и реализован в продукте Wolfram Research — Mathematica под названием Rule30CA.

Клеточный автомат – это устройство, состоящее из *n*-мерного массива ячеек и правил изменения значений ячеек. Каждая из ячеек имеет начальное состояние и изменяет свое состояние в дискретные моменты времени.

Правило, в соответствии с которым ячейка изменяет состояние, – это рекуррентная формула, в которой новое значение ячейки определяется исходя из предыдущих значений этой и соседних ячеек. В простом одномерном случае, когда массив ячеек состоит из *n* битов $a_1, a_2, \dots, a_n$, правило выглядит так:

$$a_k(t+1) = a_{k-1}(t) \oplus (a_k(t) \vee a_{k+1}(t)).$$

Случайный бит извлекается из любой ячейки.

Этот генератор показывает хорошие статистические свойства. Однако он сильно зависим от начальных значений ячеек: при неудачном выборе начального состояния, клеточный автомат порождает циклические структуры. Кроме того, для него существует успешное вскрытие с известным открытым текстом. В [17] показано, что вскрытие выполнимо на персональном компьютере с размером клеточного автомата вплоть до 500 битов.

В версии Wolfram Mathematica 10 для генерации псевдослучайных чисел можно использовать клеточный автомат, названный ExtendedCA, с более сложной формулой, которая может использовать состояния 5120 клеток одномерного клеточного автомата. График значений этого генератора представлен на рисунке 3.9.

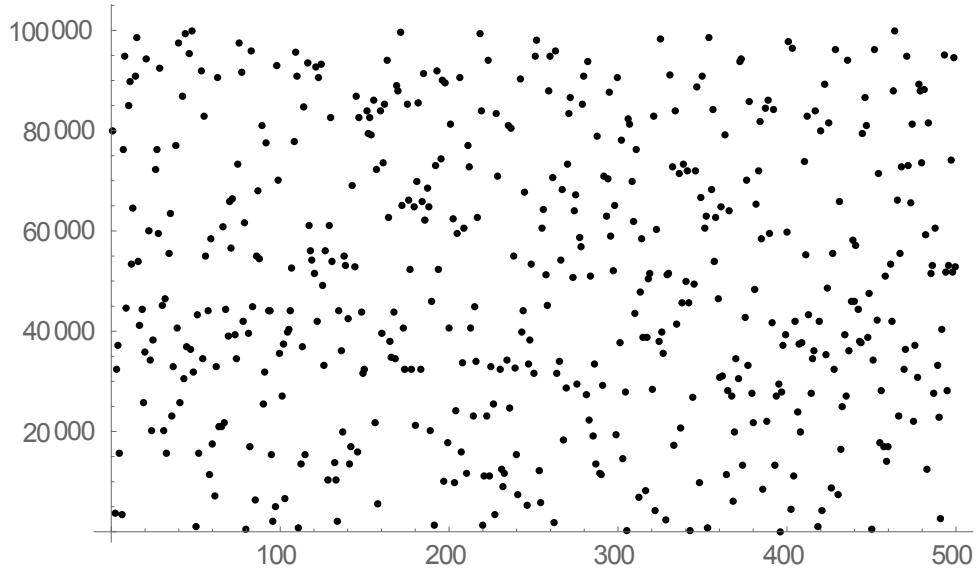

Рисунок 3.9**.** График ГПСЧ ExtendedCA.

Также необходимо отметить, что в одномерном случае выход клеточного автомата может быть сгенерирован с помощью сдвигового регистра с обратной связью той же длины и, следовательно, не дает большей безопасности [18].

3.19. Вихрь Мерсенна

Метод *Вихрь Мерсенна* [6] был предложен в 1997 году японскими учеными Макото Мацумото и Такудзи Нисимура. Метод основан на свойствах простых чисел Мерсенна и обладает рядом достоинств относительно многих других ГПСЧ. «Вихрь» – это преобразование, которое обеспечивает равномерное распределение ПСЧ.

Определение 3.7. *Числом Мерсенна* называется натуральное число M_n, определяемое формулой

$$M_n = 2^n - 1.$$

Пример. Первые 17 чисел последовательности: 1, 3, 7, 15, 31, 63, 127, 255, 511, 1023, 2047, 4095, 8191, 16 383, 32 767, 65 535, 131 071.

Часто числами Мерсенна называют числа с простыми индексами n.

Одним из важных свойств чисел Мерсенна является то, что если M_n является простым, то и n – тоже простое. Обратное в общем случае не верно, что лишает нас простого эффективного способа генерировать простые числа. Одна-

ко это свойство дает эффективный способ проверки числа на простоту, который и был положен в основу теста на простоту Люка-Лемера [1].

Существует несколько вариантов этого ГПСЧ. Мы рассмотрим наиболее распространенный, который имеет обозначение MT19937. По сути данный ГПСЧ является РСЛОС, состоящим из 624 ячеек по 32 бита. Метод Вихрь Мерсенна позволяет генерировать последовательность двоичных псевдослучайных целых *w*-битовых чисел в соответствии со следующей рекуррентной формулой

$$X_{n+p} = X_{n+q} \oplus \left(X_n^r \middle| X_{n+1}^l\right)A \quad (n = 0,1,2, \dots),$$

где *p,q,r* – целые константы, *p* – степень рекуррентности, $1 \le q \le p$;

X_n – *w*-битовое двоичное целое число;

$\left(X_n^r \middle| X_{n+1}^l\right)$ – двоичное целое число, полученное конкатенацией чисел X_n^r и X_{n+1}^l, когда первые (*w-r*) битов взяты из X_n, а последние *r* битов из X_{n+1} в том же порядке;

A – матрица размера *w×w,* состоящая из нулей и единиц, определенная посредством *a*;

XA – произведение, при вычислении которого сначала выполняют операцию $X \gg 1$ (сдвига битов на одну позицию вправо), если последний бит *X* равен 0, а затем, когда последний бит $X = 1$, вычисляют $XA = (X \gg 1) \oplus a$,

$$a = (a_{w-1}, a_{w-2}, \dots, a_0),$$
$$X = (x_{w-1}, x_{w-2}, \dots, x_0),$$
$$A = \begin{pmatrix} 0 & 1 & 0 & \dots & \dots & 0 \\ 0 & 0 & 1 & & & 0 \\ 0 & 0 & 0 & \ddots & & 0 \\ \vdots & \dots & \dots & & \ddots & \vdots \\ 0 & 0 & & & & 1 \\ a_{w-1} & a_{w-2} & \dots & \dots & \dots & a_0 \end{pmatrix}.$$

Алгоритм Вихрь Мерсенна состоит из попеременного выполнения процедур *рекурсивной генерации* и «*закалки*». Рекурсивная генерация представляет из себя РСЛОС с дополнительной рекурсивной функцией для потока выходных битов. Операция «закалки» является процедурой, усиливающей равномерность распределения на больших размерностях битовых векторов.

Шаги алгоритма.

Шаг 1а. Инициализируются значения u, h, a по формуле:

$u \coloneqq (1,0,\dots,0)$ – всего $w - r$ бит, $h \coloneqq (0,1,\dots,1)$ – всего r бит,

$a \coloneqq (a_{w-1}, a_{w-2}, \dots, a_0)$ – последняя строка матрицы A.

Шаг 1б. $X_0, X_1, \dots, X_{p-1}$ заполняются начальными значениями.

Шаг 2. Вычисляется $Y \coloneqq (y_0, y_1, \dots, y_{w-1}) := \left(X_n^r \middle| X_{n+1}^l\right)$.

Шаг 3. Вычисляется новое значение X_i:

$X_n \coloneqq X_{(n+q) \bmod p} \oplus (Y \gg 1) \oplus a$, если младший бит $y_0 = 1$;

$X_n \coloneqq X_{(n+q) \bmod p} \oplus (Y \gg 1) \oplus 0$, если младший бит $y_0 = 0$;

Шаг 4. Вычисляется $X_i T$.

$$Y \coloneqq X_n,$$
$$Y \coloneqq Y \oplus (Y \gg u),$$
$$Y \coloneqq Y \oplus \left((Y \ll s) \cdot b\right),$$
$$Y \coloneqq Y \oplus \left((Y \ll t) \cdot c\right),$$
$$Z \coloneqq Y \oplus (Y \gg l).$$

Z подается на выход, как результат.

Шаг 5. $n \coloneqq (n+1) \bmod p$. Переход на шаг 2.

Параметры алгоритма были тщательно подобраны создателями с целью достижения наилучших свойств. Параметры p и r выбраны так, что образующий многочлен – примитивный степени 19937. w выбирается по размеру стандартного машинного слова – 32 или 64 бита. Однако для 64-битной версии формула выглядит несколько иначе. Значение последней строки матрицы A выбирается случайным образом и подается на вход алгоритма. Параметры «закалки» подобраны так, чтобы дать в итоге хорошее равномерное распределение.

Параметры алгоритма Вихрь Мерсенна: $p = 624$, $w = 32$, $r = 31$, $q = 397$, $a = 2567483615$ ($9908\text{B}0\text{DF}_{16}$), $u = 11$, s=7, t=15, $l = 18$, $b = 2636928640$ ($9\text{D}2\text{C}5680_{16}$), $c = 4022730752$ ($\text{EFC}60000_{16}$).

Вихрь Мерсенна имеет огромный период, равный числу Мерсенна ($2^{19937} - 1$). Этот период достаточен для большинства возможных применений алгоритма.

Метод обеспечивает равномерное распределение генерируемых псевдослучайных чисел в 623 измерениях. Поэтому корреляция между последовательными значениями в выходной последовательности Вихря Мерсенна пренебре-

жимо мала. Метод также хорошо проходит статистические тесты на «случайность».

Однако данный ГПСЧ не предназначен для получения криптографически стойких последовательностей случайных чисел.

3.20. Рандомизация перемешиванием

Рассмотрим еще один важный класс методов основанных на комбинировании генераторов случайных чисел. Допустим, имеются две последовательности ПСЧ, сгенерированные двумя разными методами. Тогда можно, например, использовать одну последовательность для изменения порядка другой. Приведем такой алгоритм.

Пусть заданы два ГПСЧ, выдающие последовательности $\{X_n\}$ и $\{Y_n\}$ из диапазона $0, \dots m-1$. Рандомизацией будем называть метод получения новой последовательности чисел при помощи следующего алгоритма [1].

Воспользуемся таблицей $V_0, V_1, \dots, V_{k-1}$, где k – некоторое число. Вначале таблица заполняется первыми k значениями X-последовательности.

Шаг 1. Генерирование *X,Y*. Положим X и Y равными следующим членам последовательностей $\{X_n\}$ и $\{Y_n\}$ соответственно.

Шаг 2. Выбор *j*. Присвоим j значение целой части $[kY/m]$, где m – модуль, используемый в последовательности $\{Y_n\}$, т. е. j – случайная величина, определяемая $Y, 0 \le j \le k$.

Шаг 3. Замена. Выведем V_j, а затем присвоим $V_j := X$.

Есть и другие варианты этого алгоритма.

Однако методы перемешивания имеют серьезный недостаток – они изменяют порядок следования чисел, но не сами числа. Например, если последовательность бит $\{X_n\}$ не удовлетворяет критерию равномерности распределения и, например, 1 появляется чаще чем 0, то этот же недостаток будет в «перемешанной» данным алгоритмом последовательности. То же самое касается некоторых других критериев, таких как «критерий промежутков между днями рождений» или «критерий случайных блужданий», о которых речь пойдет в пятом разделе. Кроме того, перемешивание требует начального заполнения таблицы V и не позволяет начинать генерацию с любого места в периоде.

4. Криптографически стойкие ГПСЧ

Случайные числа в прикладной криптографии могут использоваться для генерации ключей, в качестве одноразовых случайных чисел в протоколах аутентификации, генерации одноразовых шифроблокнотов, в качестве соли в схемах цифровой подписи.

Определим требования к криптографически стойкому генератору псевдослучайных чисел.

4.1. Требования к КСГПСЧ

Определение 4.1. *Криптографически стойкий генератор псевдослучайных чисел* (КСГПСЧ) – генератор псевдослучайных чисел, обладающий определенными свойствами, позволяющими использовать полученные числа в криптографии. Требования для КСГПСЧ гораздо сильнее, чем для других генераторов. В частности КСГПСЧ должен обладать следующими свойствами:

1. криптографическая стойкость;
2. хорошие статистические свойства, делающие ППСЧ неотличимой от истинно случайной;
3. большой период ППСЧ;
4. реализация ГПСЧ должна быть эффективной по используемым аппаратным ресурсам.

Первое свойство более подробно будет рассмотрено далее. Второе и третье свойство аналогичны требованиям к некриптостойкому аналогу ГПСЧ. Более подробно требования к статистическим свойствам ГПСЧ будут рассмотрены в разделе 5. Четвертое свойство обусловлено необходимостью реализации на устройствах с крайне ограниченными аппаратными ресурсами – на смарт-картах, электронных ключах и т.п. Набор указанных свойств может различаться для разных задач криптографии.

Рассмотренные ранее генераторы не подходят в качестве КСГПСЧ. Для демонстрации уязвимости рассмотрим свойства криптостойкости РСЛОС.

4.1.1. Криптостойкость ГПСЧ

Рассмотрим для начала криптостойкость РСЛОС. Хотя РСЛОС быстро генерирует поток битов исходя из ключа малого размера, особенно при аппаратной реализации, они плохо подходят для целей криптографии ввиду их линейности, собственно, именно того свойства, которое придает им высокую эффективность в аппаратных средствах [14].

Обосновывая слабую криптостойкость линейных регистров, покажем, что зная длину p регистра и $2p$ последовательных битов, вышедших из РСЛОС, можно вычислить весь генерируемый поток. Заметим, что для этого нам достаточно определить значения отводов, т.е. набор a_i коэффициентов ассоциированного многочлена, поскольку стартовое состояние регистра $X_0, X_1, \ldots, X_{p-1}$ можно взять из известной нам последовательности сгенерированных битов.

Такие данные можно получить с помощью атаки с известным открытым текстом, когда мы получаем шифрограмму, соответствующую известной нам части исходного сообщения. Отметим, что функция обратной связи в этой ситуации — просто сложение значений отводов по модулю 2, действующая по формуле

$$X_j = \sum_{i=1}^{p} a_i X_{j-i} \mod 2.$$

Используя это соотношение для $j = p, p+1, \ldots, 2p-1$, получим систему p линейных уравнений с неизвестными a_i, которые нам и нужно определить. В матричной форме эта система выглядит следующим образом:

$$\begin{pmatrix} X_{p-1} & X_{p-2} & \cdots & X_1 & X_0 \\ X_p & X_{p-1} & \ldots & X_2 & X_1 \\ \vdots & & \ddots & \vdots & \\ X_{2p-3} & X_{2p-4} & \ldots & X_{p-1} & X_{p-2} \\ X_{2p-2} & X_{2p-3} & \cdots & X_p & X_{p-1} \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_{p-1} \\ a_p \end{pmatrix} = \begin{pmatrix} X_p \\ X_{p+1} \\ \vdots \\ X_{2p-2} \\ X_{2p-1} \end{pmatrix}.$$

Предположим, что мы перехватили последовательность битов 1,1,1,1,0,1,0,1,1,0,0,1,0,0,0,..., созданную 4-битовым РСЛОС. Подставляя в предыдущую систему уравнений элементы перехваченной последовательности и, решая ее в поле $GF(2)$, найдем, что многочлен, ассоциированный с РСЛОС, равен $x^4 + x + 1$. Можно сделать вывод, что поточный шифр, основанный на

единственном РСЛОС, беззащитен перед атаками с известным открытым текстом.

Мерой криптографического качества последовательности бит служит понятие линейной сложности последовательности.

Определение 4.2. Линейной сложностью бесконечной последовательности битов $X = X_1, X_2, X_3$ ... называется величина $L(X)$, равная:

- 0, если X — последовательность нулей,
- ∞, если X нельзя получить с помощью какого-нибудь РСЛОС,
- длине наименьшего РСЛОС, выдающего последовательность X в остальных случаях.

Поскольку бесконечную последовательность битов просмотреть невозможно, ограничиваются ее первыми n элементами, обозначая эту часть как X^n. Имеют место следующие свойства:

- Для любого $n \geq 1$ выполнено неравенство $0 \leq L(X^n) \leq n$.
- Если последовательность X периодична и ее период равен N, то $L(X) \leq N$.
- $L(S \oplus T) \leq L(S) + L(T)$, для любых последовательностей S и T.

Ожидаемая линейная сложность случайной двоичной последовательности X^n (именно ее мы и хотели бы видеть в качестве потока ключей) должна быть больше, чем $n/2$. С другой стороны, РСЛОС длины p выдает последовательность, чья линейная сложность удовлетворяет соотношению: $L(X^n) = p$ для всех $n \geq p$. Таким образом, линейный регистр генерирует потоки битов слишком далекие от случайных.

При генерировании потока ключей регистром сдвига с линейной обратной связью длины Z атакующий, получив не более $2p$ последовательных битов этого потока, может раскрыть регистр полностью и самостоятельно производить бегущий ключ любой нужной ему длины. Поэтому возникает необходимость найти нелинейный способ эксплуатации РСЛОС, при котором генерируемая последовательность обладает высокой линейной сложностью.

На примере РСЛОС мы увидели, что обеспечение «непредсказуемости» ГПСЧ – нетривиальная задача, и её решению посвящено много работ по информационной безопасности.

Определим наиболее общее понятие криптостойкого ГПСЧ.

Определение 4.3. Будем называть ГПСЧ *криптостойким*, если для оппонента вычислительно неразрешимы следующие задачи:

- Задача восстановления предыдущего члена последовательности (*непредсказуемость влево*) – необходимо определить элемент последовательности a_{i-1} по известным k членам последовательности $\alpha_i\alpha_{i+1}\alpha_{i+2}\dots\alpha_{i+k-1}$;
- Задача предсказания следующего члена последовательности (*непредсказуемость вправо*) – необходимо определить элемент последовательности a_{i+1} по известным k членам последовательности $a_{i-k+1}a_{i-k+2}\dots a_{i-1}a_i$;
- Задача определения ключевой информации по известному фрагменту последовательности конечной длины.

Утверждение 4.1. Если ГПСЧ является *непредсказуемым влево*, то он является криптостойким.

Следствием из этого утверждения, в частности, является то, что ГПСЧ, который проходит тест на следующий бит, является криптографически стойким генератором. Поэтому определение 4.3 эквивалентно следующему определению.

Определение 4.4. ГПСБ, который проходит тест на следующий бит называется *криптографически стойким ГПСБ* (КСГПСБ).

В качестве основного инструмента, при помощи которого реализуется КСГПСБ, используются так называемые *односторонние функции*.

Определение 4.5. Функция $f: X \to Y$ называется *односторонней функцией*, если значение $f(x)$ эффективно вычисляется за полиномиальное время, но не существует полиномиального вероятностного алгоритма, который бы мог вычислить обратную функцию $f^{-1}(x)$ для почти всех значений $y \in Im(f)$.

На практике использовать одностороннюю функцию крайне трудно, однако можно использовать специальный ее вид – *одностороннюю функцию с секретом*.

Определение 4.6. Функция $f_k: X \to Y$ называется *односторонней функцией с секретом k*, если она обладает свойствами:

- при любом k существует полиномиальный по времени алгоритм вычисления значений $f_k(x)$;
- при неизвестном k не существует полиномиального по времени алгоритма инвертирования f_k;
- существует полиномиальный по времени алгоритм инвертирования f_k при известном k.

Замечание. Определения 4.4-4.6 даны в терминах теории сложности вычислений и являются асимптотическими, так как в определении устойчивости метода к тесту на следующий бит используется понятие полиномиального по времени вероятностного алгоритма. Поэтому криптографическую стойкость мы можем трактовать с позиций невозможности вычисления некоторой функции, к которой сводится определение следующего бита последовательности.

Замечание. На существовании односторонних функций строится целое направление современной криптографии – криптография с открытым ключом. Однако не существует доказательства того, что применяемые сейчас в криптографии односторонние функции достоверно обладают вторым свойством – отсутствием полиномиального алгоритма инвертирования функции. Поэтому мы сейчас можем говорить лишь о *кандидатах* в односторонние функции. Для таких *функций-кандидатов* известно лишь то, что они эквивалентны некоторым хорошо изученным вычислительно сложным математическим задачам. Таким образом, на практике второе свойство заменяется на более слабое условие: при неизвестном k, *вероятно* не существует полиномиального по времени алгоритма вычисления функции f_k^{-1}.

Как правило, КСГПСЧ использует некоторую последовательность случайных чисел, не обладающую криптостойкостью, преобразуя её при помощи некоторой односторонней функции с секретом. В результате последовательность чисел «наделяется» свойством криптостойкости. Так как математический аппарат криптографии в значительной мере опирается на использование односторонних функций, мы можем использовать весь спектр существующих методов криптографии и для решения нашей задачи – генерации криптостойкой ППСЧ.

Существует три больших класса алгоритмов, используемых в КСГПСЧ:

1. методы на основе криптографических алгоритмов;

2. методы на основе односторонних функций (математически сложных задач);
3. специальные реализации.

В первом случае генерация ПСЧ может быть реализована одним из следующих способов:

- применением безопасного блочного шифра в режиме счетчика (гаммирования) к некоторому случайному ключу;
- применением криптографически стойкой хэш-функции к исходному секретному случайному числу;
- применением потоковых шифров, которые, в свою очередь, сами работают на основе генерации псевдослучайных бит.

Во втором классе алгоритмов для генерации используются односторонние функции с ключом, в качестве которых может служить функция возведении в квадрат, функция экспоненцирования, функция умножения простых чисел в конечном поле.

Третий класс алгоритмов основан на использовании стохастических методов преобразования чисел [49, с.91-117]. Рассмотрим эти классы алгоритмов подробнее.

4.2. Безопасный блочный шифр

Определение 4.7. *Блочный шифр* – разновидность шифра, основными особенностями которого являются:

- шифрование исходного текста блоками (например, по n-бит);
- содержимое каждого блока никак не влияет на результат шифрования других блоков.

Формально, блочный шифр можно представить функцией

$$E: \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n.$$

В этих обозначениях функция имеет два входных параметра: первый k-битовая строка, называемая ключом и обозначаемая $K \in \{0,1\}^k$, и n-битовая строка блока сообщения, обозначаемая $M \in \{0,1\}^n$. Результат функции называется шифротекстом и обозначается обычно $C \in \{0,1\}^n$.

Для конкретного ключа K, функция блочного шифра будет обозначаться

$$E_K(M) = C.$$

Для каждого определенного ключа K, функция блочного шифра должна быть взаимно однозначным отображением. Это значит, что E_K всегда имеет обратную функцию E_K^{-1} такую, что $E_K\left(E_K^{-1}(C)\right) = C$ и $E_K^{-1}\left(E_K(M)\right) = M$ для любых $M, C \in \{0,1\}^n$.

Определение 4.8. *Зашифрованием* будем называть процесс преобразования алгоритмом выбранного режима шифрования открытого текста сообщения в шифротекст. Обратный же процесс *расшифрования* алгоритмом режима шифрования выполняется для восстановления открытого текста сообщения из шифротекста.

Функция шифрования должна обладать рядом свойств. В частности, эта функция должна быть односторонней функцией с ключом. То есть функция $E_K^{-1}(C)$ должна быть трудно вычислима без знания значения K. Под сложностью вычисления мы понимаем невозможность за приемлемое время (доступными ресурсами) вычислить эту функцию.

Стандартный способ применения блочного шифра:

- выбор некоторого ключа K, который известен двум сторонам информационного обмена, но держится в секрете от остальных;
- генерация шифротекста путем применения функции шифрования к сообщению, перед его отправкой партнеру $C = E_K(M)$;
- отправка шифротекста партнеру;
- получение и расшифровка партнером шифротекста путем применения обратной функции $M = E_K^{-1}(C)$.

Описание конкретных алгоритмов блочных шифров, а также теоретического обоснования их криптостойкости выходит за рамки нашей книги, однако любой интересующийся может обратиться к многочисленным источникам и опубликованным стандартам, подробно описывающим алгоритмы шифрования [3, 10, 14, 15, 29]. Мы же остановимся на некоторых способах применения блочных шифров для генерации псевдослучайных чисел.

Основное назначение блочного шифра – сокрытие от оппонента смысла передаваемого сообщения. Применение шифрования для генерации ПСЧ, требует некоторых дополнительных пояснений.

Определение 4.9. *Режим шифрования блочного шифра* – это такой алгоритм применения блочного шифра, который при отправке сообщения позволяет преобразовывать открытый текст в шифротекст и после передачи этого шифротекста по открытому каналу однозначно восстановить первоначальный открытый текст.

В документе [4], изданном NIST США и озаглавленном "Рекомендации для режимов шифрования с блочным шифром" (NIST SP800-38A), описаны пять режимов шифрования алгоритмом AES. Особый интерес для нас представляет режим счетчика (Counter mode – CTR). Этот режим шифрования может быть применен для генерации последовательности псевдослучайных чисел.

Определение 4.10. *Режим счетчика блочного шифра* (CTR mode) – такой режим шифрования входной последовательности, при котором на вход соответствующего алгоритма блочного шифрования подаются значения, так называемого счетчика, накопленного с момента старта. Меняя значение счетчика, алгоритм блочного шифрования образует строку битов, которая используется в качестве бегущего ключа шифра Вернама, т.е. к бегущему ключу и блокам исходного сообщения применяются операции побитового сложения по модулю 2.

По сути, режим счетчика делает из блочного шифра потоковый.

Режим CTR предусматривает следующие операции:

$$C_i = P_i \oplus E_k(T_i);\ \ i = 1,2,\dots,m,$$

где i – номер блока, C_i- зашифрованный текст, P_i – блок сообщения, E_k – алгоритм шифрования (DES, AES), T_i – значение счетчика.

Значения счетчика должны быть уникальны для каждого блока шифруемого текста. Для этого используют некоторые приемы. Один из вариантов получения нового значения счетчика:

$$L_{i+1} = (L_i + 1) \bmod 2^m,$$

$$T_i = M_i | L_{i+1},$$

где | - функция конкатенации; L_i – младшие m битов; M_i – старшие (b-m) битов. Уникальность значений счетчика обеспечивается для всех блоков сообщения при условии, что $n \leq 2^m$, где n – количество блоков, на которое разбивается сообщение.

Кроме того, отдельного подхода требует проблема начального значения счетчика. Способы её решения также описаны в стандарте NIST SP800-38A.

Использование данного режима шифрования для нас интересно тем, что при условии корректной инициализации и приращения счетчика, сгенерированные значения $E_k(T_i)$ могут быть использованы в качестве псевдослучайных чисел.

Очевидно, что период последовательности будет не больше, чем 2^n для n-битного блочного шифра. Также очевидно, что безопасность такой схемы полностью зависит от секретности ключа.

Аналогично, в российском стандарте ГОСТ 28147-89 описан режим работы *гаммирования* и режим *гаммирования с обратной связью*, который может быть использован для генерации последовательности ПСЧ.

4.3. ANSI X9.17

Следующий алгоритм одобрен FIPS и утвержден стандартом ANSI X9.17 для использования в целях генерации псевдослучайных ключей и инициализационных векторов в алгоритме шифрования DES. Так как алгоритм DES сейчас уже не считается безопасным, то ГПСЧ на его основе также не может считаться безопасным. Однако этот ГПСЧ получил большое распространение, в частности, применяется в PGP[1].

Обозначим E_k функцию шифрования алгоритмом Triple DES в режиме двухключевой EDE.

Описание алгоритма.

На вход подаются: секретный начальный 64-битовый вектор s, целое m, и секретный 112-битный ключ для DES шифрования k.

На выходе: m псевдослучайных чисел $x_1, x_2, \dots, x_m$.

1. Вычислить промежуточное значение $I = E_k(D)$, где D – 64-битное представление времени с максимально возможной точностью.
2. $For\ i = 1\ to\ m$
   a. $x_i \leftarrow E_k(I \oplus s)$.
   b. $s \leftarrow E_k(x_i \oplus I)$.
3. Вернуть $(x_1, x_2, \dots, x_m)$.

[1] PGP – популярная программа, позволяющая выполнять шифрование и простановку/проверку цифровой подписи сообщений, файлов и другой информации, представленной в электронном виде. Официальный сайт - http://www.pgpi.org.

Любая строка битов x_i может быть использована в качестве инициализационного вектора (*IV*) для одного из режимов шифрования DES.

4.4. FIPS 186. Алгоритм генерации секретного ключа для ЭЦП.

Данный алгоритм представлен в рекомендованных методах для генерации псевдослучайных параметров электронной цифровой подписи. Этот алгоритм генерирует секретный ключ a, используя для этого начальный вектор s, который должен быть сгенерирован случайным образом и преобразован хэш-функцией SHA-1.

Описание алгоритма.

На вход подаются: целое *m* и 160-битное простое число q.

На выходе: *m* псевдослучайных чисел $a_1, a_2, \dots, a_m$ в интервале $[0, \mathrm{q} - 1]$, которые могут быть использованы в качестве секретных ключей для ЭЦП.

1. Выбрать 160-битное b.
2. Определить 160-битную строку t=0x67452301 EFCDAB89 98BADCFE 10325476 C3D2E1F0.
3. $For\ i = 1\ to\ m$
   a. $y_i \leftarrow 0$.
   b. $z_i \leftarrow (s + y_i) \bmod 2^b$.
   c. $a_i \leftarrow G(t, z_i) \bmod q$.
   d. $s \leftarrow (1 + s + a_i) \bmod 2^b$.
4. Вернуть $(a_1, a_2, \dots, a_m)$.

В качестве функции G можно выбрать либо функцию SHA-1 (п. 3.3.5), либо функцию алгоритма DES.

Примечание. На шаге 3.а для усиления криптостойкости алгоритма, переменную y_i можно инициализировать не нулем, а случайно сгенерированным значением.

4.5. FIPS 186. Алгоритм генерации секретного числа сообщения для ЭЦП

На входе: целое *m* и 160-битное простое число b.

На выходе: *m* псевдослучайных чисел $k_1, k_2, \dots, k_m$ в интервале $[0, q-1]$, которые могут быть использованы как секретные случайные числа сообщения в ЭЦП.

1. Выбрать 160-битное *b*.
2. Определить 160-битную строку *t*=0xEFCDAB89 98BADCFE 10325476 C3D2E1F0 67452301.
3. $For\ i = 1\ to\ m$
   a. $k_i \leftarrow G(t, s)\ mod\ q$.
   b. $s \leftarrow (1 + s + k_i)\ mod\ 2^b$.
4. Вернуть $(k_1, k_2, \dots, k_m)$.

В качестве функции G можно выбрать либо функцию SHA-1, либо функцию DES.

4.5.1. Алгоритм генерации ПСЧ с использованием SHA-1

На входе: 160-битная строка *t* и *b*-битная строка с, $160 \le b \le 512$.

На выходе: 160-битная строка $G(t, c)$

1. Разбить строку *t* на пять 32-битных блоков $t = H_1|H_2|H_3|H_4|H_5$.
2. Дополнить c нулями до 512-битной длины: $X \leftarrow c|0^{512-b}$.
3. Разбить X на 16 32-битных слов $M_0 M_1 \dots M_{15}$, и установить $m \leftarrow 1$.
4. Вычисляем новые значения $(H_1, H_2, H_3, H_4, H_5)$ алгоритмом SHA-1.
5. Вычисляем $G(t, c) = H_1|H_2|H_3|H_4|H_5$.

Более подробно, про функцию SHA-1 будет рассказано в п. 4.6.5.

4.5.2. Алгоритм вычисления односторонней функции с использованием DES

На входе: две 160-битных строки *t* и *c*.

На выходе: 160-битная строка $G(t, c)$

1. Разбить *t* на 32-битные блоки: $t = t_0|t_1|t_2|t_3|t_4$.
2. Разбить с на 32-битные блоки: $c = c_0|c_1|c_2|c_3|c_4$.
3. $For\ i = 0\ to\ 4\ do\ \ x_i \leftarrow t_i \oplus c_i$.
4. $For\ i = 0\ to\ 4\ do$
   a. $b_1 \leftarrow c_{(i+4)\ mod\ 5}, b_2 \leftarrow c_{(i+3)\ mod\ 5}$.
   b. $a_1 \leftarrow x_i,\ \ a_2 \leftarrow x_{(i+1)\ mod\ 5} \oplus x_{(i+4)\ mod\ 5}$.
   c. $A \leftarrow a_1|a_2, B \leftarrow b_1'|b_2$, где b_1' - 24 последних значащих бита b_1.

5. Использовать DES с ключом B для шифрования A: $y_i \leftarrow DES_B(A)$.
6. Разбить y_i на два 32-битных блока:$y_i = L_i|R_i$.
7. $For\ i = 0\ to\ 4\ do$
   a. $z_i \leftarrow L_i \oplus R_{(i+2)\ mod\ 5} \oplus L_{(i+3)\ mod\ 5}$.
8. Выход: $G(t,c) = z_0|z_1|z_2|z_3|z_4$.

4.6. Криптографически стойкая хэш-функция

Рассмотрим однонаправленную функцию $H(M)$

$$H: V^* \rightarrow V_n,$$

где V^* – множество всех двоичных векторов конечной размерности, включая пустую строку, а V_n – множество всех n-мерных двоичных векторов, где n – целое неотрицательное число. Будучи примененной к сообщению M произвольной длины, она вычисляет значение h фиксированной длины m:

$$h = H(M).$$

Очевидно, таких функций много. Криптографические применения требуют от односторонней функции наличия дополнительных свойств:

- Зная M, легко вычислить h;
- Зная H, трудно вычислить M, для которого $H(M) = h$;
- Зная M, трудно вычислить другое сообщение M', для которого $H(M) = H(M')$.

Функция, удовлетворяющая этим свойствам, будет называться криптографически стойкой хэш-функцией. Приведем формальное определение.

Определение 4.10. Хэш-функция H является криптографически стойкой, если она удовлетворяет трем требованиям:

1. Необратимость или стойкость к восстановлению прообраза: для заданного значения хэш-функции h должно быть вычислительно невозможно найти блок данных X, для которого $H(X) = h$.
2. Стойкость к коллизиям первого рода или к восстановлению вторых прообразов: для заданного сообщения M должно быть вычислительно невозможно подобрать другое сообщение N, для которого $H(N)=H(M)$.
3. Стойкость к коллизиям второго рода: должно быть вычислительно невозможно подобрать пару сообщений (M,M'), имеющих одинаковое значение хэш-функции.

Значение хэш-функции называют хэш-кодом или хэш-значением. А процесс получения хэш-кода называют хэшированием сообщения.

Примечание. Не стоит путать криптостойкое хэширование сообщения с хэшированием таблиц, используемым в программировании. Дело в том, что для вычислений при хэшировании таблиц (создания словарей) в программах используют односторонние функции, в которых требования стойкости к коллизиям первого и второго рода заменяются требованием «хорошего» распределения хэш-значений. Это требование означает необходимость минимизации вероятности совпадения хэш-значений для различных исходных сообщений, но не предъявляет требований к сложности нахождения таких сообщений злоумышленником. Для задач криптографии этого требования к функции недостаточно.

Примечание. Во многих приложениях от хэш-функции требуется только быстрая вычислимость и первое свойство (стойкость к восстановлению прообраза). Функции, которые не обладают вторым и третьим свойством из определения (стойкостью к коллизиям), будем называть криптографически нестойкими хэш-функциями или просто не криптографическими хэш-функциями.

4.6.1. Использование хэш-функции в ГПСЧ

Использование хэш-функции для генерации КСПСЧ подобно использованию блочного шифра для этой же цели. Выбрав некоторое инициализирующее натуральное число S_1, мы последовательно применяем преобразование $S_{i+1} = H(S_i)$. Алгоритм может быть модифицирован, но главная его уязвимость останется – безопасность полностью зависит от инициализирующего значения. Узнав инициализирующее значение и алгоритм ГПСЧ, злоумышленник может построить всю последовательность ПСЧ.

Кроме того, для построения криптостойких ГПСЧ рекомендуется использовать хэш-функцию для генерации начального вектора, реинициализации, генерации внутреннего состояния и генерации очередного числа последовательности (подробно описано в п. 2.4). Все эти требования увеличивают надежность алгоритма и снижают вероятность успешной атаки на ГПСЧ.

Рассмотрим некоторые хэш-функции.

4.6.2. CRC32 (Cyclic redundancy check)

CRC32 (Cyclic redundancy check/code – циклический избыточный код) – не является криптографически стойкой хэш-функцией, так как не обладает устойчивостью к коллизиям. Тем не менее, эта функция широко применяется в различных приложениях для защиты от случайных (ненамеренных) изменений данных, обнаружения ошибок при передаче данных и т.п.

Подробно алгоритм вычисления CRC32 описан в стандарте CRC32-IEEE 802.3 [21]. Изложим коротко суть алгоритма. Любое сообщение, состоящее из последовательности n битов a_i, может быть представлено как двоичный многочлен степени n

$$M(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0.$$

Для вычисления CRC сообщения нам понадобится еще один многочлен, называемый *порождающим многочленом* и обозначаемый $G(x)$. $G(x)$ должен иметь степень больше чем ноль и меньше чем n. Кроме того, $G(x)$ должен быть неприводимым многочленом над $GF(2)$ и иметь ненулевой коэффициент при x_0.

Пример. $G(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$.

Значение контрольной суммы в алгоритме с порождающим многочленом $G(x)$ степени m определяется как битовая последовательность длины m, представляющая многочлен $R(x)$, получаемый в остатке при делении многочлена $M(x)$ на $G(x)$. Для улучшения качества хэширования коротких последовательностей $M(x)$ умножают на x^m, что является сдвигом регистра на m битов влево:

$$R(x) = M(x) \cdot x^m \bmod G(x).$$

Процесс вычисления строится итеративно: сообщение разбивается на блоки длиной 128 (или более) битов. На каждой итерации вычисляется значение CRC от результата побитового сложения нового блока сообщения с результатом вычисления CRC на предыдущей итерации. Результат работы алгоритма на последнем блоке сообщения и является выходом алгоритма.

4.6.3. MD4

MD4 – алгоритм хэширования, разработанный Рональдом Л. Ривестом из RSA Data Security, Inc. Алгоритм описан в RFC 1320. Хэш-значение представляет шестнадцатеричное число из 32 4-битовых символов. Этот алгоритм не является криптографически стойким, однако на базе этого алгоритма было разработано целое семейство других более сложных алгоритмов хэширования, – MD5, SHA-1, RIPEMD-160, SHA-256, SHA-512, SHA-384. Поэтому рассмотрим MD4 более подробно, а с его усложненными версиями можно ознакомиться в специализированной литературе.

Описание алгоритма MD4 [14, 22].

Исходные данные. На вход алгоритма поступают блоки сообщения размером 512 битов, разбитые по 16 32-битовых блоков X_i. На выходе алгоритма получаем хэш-значение длиной 128 битов.

В MD4 участвуют три поразрядные функции от трех 32-битовых переменных:

$$f(u,v,w) = (u \wedge v) \vee (\overline{u} \wedge w),$$

$$g(u,v,w) = (u \wedge v) \vee (u \wedge w) \vee (v \wedge w),$$

$$h(u,v,w) = u \oplus v \oplus w.$$

Выравнивание сообщения. Сначала сообщение дополняется так, чтобы его длина была короче числа, кратного 512 на 64 бита – добавляется 1, за которой идут одни 0. Оставшиеся 64 бита до кратной 512 длины сообщения заполняется значением длины сообщения (истинной, до выравнивания).

Инициализация. На протяжении всего алгоритма мы следим за текущим хэш-состоянием (H_1, H_2, H_3, H_4) 32-битовых переменных, начальные значения которых изначально равны

$$H_1 = 0\text{x}67452301, H_2 = 0\text{x}EFCDAB89, H_3 = 0\text{x}98BADCFE, H_4 = 0\text{x}10325476.$$

Они называются переменными сцепления. Кроме того, для каждого раунда (всего раундов три), существуют свои константы (y_i, z_i, s_i):

$$y_j = \begin{cases} 0, & 0 \le j \le 15, \\ 0\text{x}5A827999, & 16 \le j \le 31, \\ 0\text{x}6ED9EBA1, & 32 \le j \le 47. \end{cases}$$

$$z_{0\ldots15} = [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15],$$
$$z_{16\ldots31} = [0,4,8,12,1,5,9,13,2,6,10,14,3,7,11,15],$$
$$z_{32\ldots47} = [0,8,4,12,2,10,6,14,1,9,5,13,3,11,7,15],$$
$$s_{0\ldots15} = [3,7,11,19,3,7,11,19,3,7,11,19,3,7,11,19],$$
$$s_{16\ldots31} = [3,5,9,13,3,5,9,13,3,5,9,13,3,5,9,13],$$
$$s_{32\ldots47} = [3,9,11,15,3,9,11,15,3,9,11,15,3,9,11,15].$$

Вычисления в цикле. Поток данных состоит из 16 одновременно загружаемых слов в массив X_j $(0 \le j < 16)$. Затем мы выполняем следующие преобразования над каждым из 16 слов, внесенных в поток данных:

1. $(A, B, C, D) = (H_1, H_2, H_3, H_4)$.
2. Выполнить первый раунд.
3. Выполнить второй раунд.
4. Выполнить третий раунд.
5. $(H_1, H_2, H_3, H_4) = (H_1 + A, H_2 + B, H_3 + C, H_4 + D)$.

После считывания всех данных выходные данные представляют собой конкатенацию окончательных значений переменных H_1, H_2, H_3, H_4.

Описание раундов.

Раунд 1

$$FOR\ j = 0\ to\ 15$$
$$\{$$
$$1.\ t = A + f(B, C, D) + X_{z_j} + y_j.$$
$$2.\ (A, B, C, D) = \left(D, t \lll s_j, B, C\right).$$
$$\}$$

Раунд 2

$$FOR\ j = 16\ to\ 31$$
$$\{$$
$$1.\ t = A + g(B, C, D) + X_{z_j} + y_j.$$

$2. (A, B, C, D) = \big(D, t \lll s_j, B, C\big).$

}

Раунд 3

$FOR\ j = 32\ to\ 47$

{

$1. t = A + h(B, C, D) + X_{z_j} + y_j.$

$2. (A, B, C, D) = \big(D, t \lll s_j, B, C\big).$

}

Здесь символ ⋘ обозначает циклический сдвиг влево.

4.6.4. MD5

MD5 является улучшенной версией MD4. Подробное описание алгоритма дано в RFC 1321[22]. Схема вычислений усложнена относительно MD4, однако это не сделало шифр устойчивым к атакам на поиск коллизий. Так, в 2004 году китайские исследователи Ван Сяоюнь (Wang Xiaoyun), Фэн Дэнго (Feng Dengguo), Лай Сюэцзя (Lai Xuejia) и Юй Хунбо (Yu Hongbo) объявили об обнаруженной ими уязвимости в алгоритме, позволяющей за небольшое время находить коллизии [24].

В 2005 году Ван Сяоюнь и Юй Хунбо из университета Шаньдуна в Китае опубликовали алгоритм, который может найти две различные последовательности в 128 байт, которые дают одинаковый MD5-хэш.

В 2006 году чешский исследователь Властимил Клима опубликовал алгоритм, позволяющий находить коллизии на обычном компьютере с любым начальным вектором (*A,B,C,D*) при помощи метода, названного им «туннелирование» [45].

4.6.5. SHA-1

SHA (Secure Hash Algorithm) – алгоритм безопасного хэширования разработан в 1995 году, NIST совместно с АНБ США в рамках стандарта Secure Hash Standart (SHS) для совместного использования с электронной цифровой подписью (DSA) . Алгоритм преобразует входное сообщение длиной не более 2^{64} битов в 160-битовый результат. Принципы, лежащие в основе SHA, аналогичны

использованным Рональдом Л. Ривестом при проектировании алгоритма MD4. Подробное описание стандарта дано в FIPS PUB 180-1.

Краткое описание алгоритма.

Исходные данные. Входом алгоритма является блок длиной 512 бит и выход предыдущего блока.

Определяются четыре нелинейные операции и четыре константы:

$$F_t(X,Y,Z) = \begin{cases} (X \wedge Y) \vee \big((\neg X) \wedge Z\big), & 0 \le t \le 19, \\ X \oplus Y \oplus Z, & 20 \le t \le 39, \\ (X \wedge Y) \vee (X \wedge Z) \vee (Y \wedge Z), & 40 \le t \le 59, \\ X \oplus Y \oplus Z, & 60 \le t \le 79. \end{cases}$$

$$K_t = \begin{cases} 0\text{x}5A827999, & 0 \le t \le 19, \\ 0\text{x}6Ed9EBA1, & 20 \le t \le 39, \\ 0\text{x}8FLBBCDC, & 40 \le t \le 59, \\ 0\text{x}CA62C1D6, & 60 \le t \le 79. \end{cases}$$

Выравнивание сообщения. Сначала сообщение выравнивается так, чтобы его длина была кратна 512 битам. Это делается аналогично алгоритму MD4: в конец сообщения добавляется 1, а затем, нули, вплоть до длины 448 битов. После этого добавляются 64 бита, содержащие длину исходного сообщения.

Инициализация. Инициализируется пять 32-битовых переменных буфера:

$$H_1 = 0x67452301, H_2 = 0xEFCDAB89, H_3 = 0x98BADCFE,$$
$$H_4 = 0x10325476, H_5 = 0xC3D2e1F0.$$

Вычисления в цикле. В главном цикле по порядку обрабатывается очередной 512-битовый блок сообщения вплоть до последнего.

Сначала значения переменных буфера копируются во вспомогательные переменные. Потом в итерации цикла проводятся 4 раунда преобразований над вспомогательными переменными. Каждый раунд состоит из 20 операций. Каждая операция – нелинейное преобразование над тремя из пяти переменных A, B, C, D, E. Затем производится сдвиг и побитовое сложение аналогично MD4.

1. $(A, B, C, D, E) = (H_1, H_2, H_3, H_4, H_5)$.
2. Преобразование блока сообщения из 16 32-битовых слов $(M_0, M_1, \dots, M_{15})$ в 80 32-битных слов $(W_0, W_1, \dots, W_{79})$:

$$W_t = M_t, 0 \le t \le 15,$$
$$W_t = (W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16} \lll 1, 16 \le t \le 79.$$

Здесь символ $\lll$ обозначает циклический сдвиг влево.

3. Если t – это номер операции (от 1 до 80), W_t t-й подблок расширенного сообщения, а $\lll s$ – это циклический сдвиг влево на s битов, то главный цикл выглядит следующим образом:

$FOR\ t = 0\ to\ 79$
{
$TEMP = (A \lll 5) + F_t(B, C, D) + E + W_t + K_t$
$E = D$
$D = C$
$C = B \lll 30$
$B = A$
$A = TEMP$
}

4. $(H_1, H_2, H_3, H_4, H_5) = (H_1 + A, H_2 + B, H_3 + C, H_4 + D, H_5 + E)$.

После этого алгоритм продолжается для следующего блока. Результатом работы алгоритма будет вычисленные $(H_1, H_2, H_3, H_4, H_5)$ для последнего блока.

Помимо более длинного – 160-битового хэш-значения, главным отличием SHA-1 от MD4 является введение расширяющего преобразования и добавление выхода предыдущего шага в следующий. что по словам Рона Ривеста, обеспечивает более быстрый лавинный[1] эффект. На лавинный эффект также оказывает влияние циклический сдвиг влево. Кроме того, в алгоритме появились константы K_t.

Более длинное 160-битовое хэш-значение делает алгоритм устойчивым к атакам грубой силой. В феврале 2005 года Сяоюнь Ван, Ицюнь Лиза Инь и Хунбо Юй (Xiaoyun Wang, Yiqun Lisa Yin, Hongbo Yu) представили атаку на полноценный SHA-1, которая требует менее 2^{69} операций. Несмотря на эффективность предложенного метода, на практике эта атака не осуществима.

[1] Лавинный эффект – термин, впервые введенный Хорстом Файстелем, означающий, что изменение значения любого бита текста, при применении к нему криптографического алгоритма, ведет к изменению в среднем половины значений выходных битов шифротекста.

4.6.6. ГОСТ Р 34.11-94

Для реализации функции хэширования используется блочный алгоритм ГОСТ 28147-89, хотя допускается использование любого криптостойкого блочного алгоритма шифрования с 64-битовым блоком и 256-битовым ключом. Результатом функции является значение длиной 256 битов.

Описание алгоритма.

На входе: 256-битовые значения M_i, H_i.

На выходе: 256-битовое значение H_{i+1}.

1. При помощи линейного смешивания M_i, H_{i-1} и некоторых констант генерируется четыре ключа шифрования ГОСТ.
2. Каждый ключ используется для шифрования разных 64-битовых блоков H_{i-1} в режиме гаммирования. Полученные 256 бит сохраняются в переменной *S*.
3. H_i является сложной, хотя и линейной функцией S, M_i и H_{i-1}.

Хэш-значение последнего блока сообщения не является его окончательным хэш-значением. На деле используются переменные сцепления: H_n – это хэш-значение последнего блока, *Z* – это побитовая сумма по модулю 2 всех блоков сообщения, а *L* – длина сообщения. С использованием этих переменных и дополненного последнего блока *M'*, окончательное хэш-значение равно

$$H = f\left(Z \oplus M', f\left(L, f\left(M', H_n\right)\right)\right).$$

4.6.7. ГОСТ Р 34.11-2012

ГОСТ Р.34.11-2012 – действующий национальный российский стандарт, определяющий алгоритм и процедуры вычисления хэш-функции для любой последовательности двоичных символов, которые применяются в криптографических методах обработки и защиты информации, в том числе для реализации процедур обеспечения целостности, аутентичности, электронной цифровой подписи (ЭЦП) при передаче, обработке и хранении информации в автоматизированных системах. Этот алгоритм имеет также неофициальное название «Стрибог», хотя в тексте стандарта оно нигде не упоминается.

Основное отличие новой хэш-функции от старой – функция сжатия. В ГОСТ Р 34.11–2012 используется функции сжатия в основе которой лежат три

преобразования: нелинейное биективное преобразование (обозначается *S*), перестановка байт (обозначается *P*), линейное преобразование (обозначается *L*). В ГОСТ Р 34.11–94 используется функция сжатия, основанная на симметричном блочном шифре ГОСТ Р 28147-89, также эта функция использует операции перемешивания. Кроме того в новом стандарте приводятся значения инициализационного вектора (для 256-битового и 512-битового варианта).

Для начала опишем преобразования, используемые в вычислениях:

π – подстановка, преобразующая целое восьмибитовое число в восьмибитовое число по заданной таблице [25, 26].

τ – перестановка, сопоставляющая 64-битовому число 64-битовое число.

l – линейное преобразование множества двоичных векторов V_{64}, заданное умножением справа на матрицу A над полем $GF(2)$, которая приведена в стандарте.

$$X[k]: V_{512} \to V_{512}, \qquad X[k](a) = k \oplus a, \qquad k, a \in V_{512};$$

$$S: V_{512} \to V_{512}, \quad S(a) = S(a_{63} \| \cdots \| a_0) = \pi(a_{63}) \| \cdots \| \pi(a_0);$$

$$P: V_{512} \to V_{512}, \quad P(a) = P(a_{63} \| \cdots \| a_0) = a_{\tau(63)} \| \cdots \| a_{\tau(0)};$$

$$L: V_{512} \to V_{512}, \qquad L(a) = L(a_7 \| \cdots \| a_0) = l(a_7) \| \cdots \| l(a_0),$$

$$a = a_{63} \| \cdots \| a_0 \in V_{512}, \qquad a_i \in V_8, \qquad i = 0, \dots, 63;$$

Описание алгоритма

Исходные данные: подлежащее хэшированию сообщение $M \in V^*$ и $IV \in V_{512}$ – инициализационный вектор хэширования.

На выходе: 512-битовый хэш-код H(M).

Значение хэш-кода сообщения M вычисляется с использованием итерационной процедуры. На каждой итерации вычисления хэш-кода используется функция сжатия

$$g_N: V_{512} \times V_{512} \to V_{512}, N \in V_{512},$$

значение которой вычисляется по формуле:

$$g_N(h, m) = E(LPS(h \oplus N), m) \oplus h \oplus m,$$

где $E(K, m) = X[K_{13}]LPSX[K_{12}] \dots LPSX[K_2]LPSX[K_1](m)$.

Значения $K_i \in V_{512}$, $\mathrm{i} = 1, \dots, 13$ вычисляются следующим образом:

$$K_1 = K;$$

$$K_i = LPS(K_{i-1} \oplus C_{i-1}), i = 2, \dots, 13.$$

Операцию сложения в кольце Z_{2^n} ,будем обозначать символом $\boxplus$.

Алгоритм вычисления функции H состоит из трех этапов.

1. **Этап 1.** Присвоить начальные значения величин:
   1.1. $h := IV$;
   1.2. $N := 0^{512} \in V_{512}$;
   1.3. $\Sigma := 0^{512} \in V_{512}$;
   1.4. Перейти к этапу 2.
2. **Этап 2.**
   2.1. Проверить условие $|M| < 512$. Если выполняется, то переход к этапу 3. Иначе вычислить:
   2.2. Вычислить $m \in V_{512}$ такой, что $M = M' \parallel m$.
   2.3. $h := g_N(h, m)$;
   2.4. $N := N \boxplus 512$;
   2.5. $\Sigma := \Sigma \boxplus m$;
   2.6. $M := M'$;
   2.7. Перейти к шагу 2.1.
3. **Этап 3.**
   3.1. $m := 0^{511-|M|} \parallel 1 \parallel M$;
   3.2. $h := g_N(h, m)$;
   3.3. $N := N \boxplus |M|$;
   3.4. $\Sigma := \Sigma \boxplus m$;
   3.5. $h := g_0(h, N)$;
   3.6. $h := g_0(h, \Sigma)$;
   3.7. Конец работы алгоритма.

Значение величины h, полученное на шаге 3.6, является значением функции хэширования $H(M)$.

4.7. ГПСЧ использующие алгоритмы потокового шифра

Большинство потоковых шифров работают на основе генерации псевдослучайного потока бит, которые некоторым образом комбинируются с битами

открытого текста. Запуск такого шифра на последовательности натуральных чисел даст новую псевдослучайную последовательность, возможно даже с более длинным периодом. Такой метод безопасен только если в самом потоковом шифре используется надежный КСГПСЧ (что не всегда так). Часто для создания потоковых чисел используется РСЛОС, что автоматически делает такой шифр криптографически нестойким. Опять же, начальное состояние счетчика должно оставаться секретным.

4.7.1. Алгоритм RC4

Алгоритм RC4 [14, 10] назван по имени его создателя – Ron's Cipher (Рон Ривест). Алгоритм был разработан компанией RSA Data Security, Inc. в 1987 году. Являясь потоковым шифром, в основе которого генератор псевдослучайных чисел, RC4 широко используется в различных криптографических протоколах. Достоинством алгоритма является высокая скорость работы и переменный размер ключа.

Описание алгоритма.

1. Инициализация S_i, $i = 0,1, \dots ,255$.
   a) $for\ i\ =\ 0\ to\ 255: S_i = i$;
   b) $j = 0$;
   c) $for\ i = 0\ to\ 255: j = (j + S_i + K_i)\ mod\ 256; Swap(S_i, S_j)$
2. $i = 0, j = 0$.
3. Итерация алгоритма:
   a) $i = (i + 1) \bmod 256$;
   b) $j = (j + S_i) \bmod 256$;
   c) $Swap(S_i, S_j)$;
   d) $t = (S_i + S_j) \bmod 256$;
   e) $K = S_t$;

Стойкость алгоритма основана на следующем наблюдении: даже если оппонент узнал ключ K и номер шага i, он может вычислить всего лишь значение

S_t, но не всё внутреннее состояние массива. Это следует из того, что оппонент не в состоянии определить значение переменной *t*, не зная j, S_i или S_j.

Каждый шаг алгоритма усиливает его стойкость:

- шаг a обеспечивает однократное использование элементов массива;
- шаг b обеспечивает нелинейную зависимость выхода от массива;
- шаг c изменяет массив в процессе итераций;
- шаг d скрывает внутреннее состояние массива от анализа.

Алгоритм RC4 входит в различное коммерческое программное обеспечение: Lotus Notes, AOCE (Apple Computer), Oracle Secure SQL.

4.7.2. ГПСЧ Шамира и RSA

Алгоритмы ГПСЧ на основе сложных математических задач используют сложность решения некоторых задач для получения псевдослучайных чисел. защищенных от криптоанализа. Другими словами, криптограф, создающий ГПСЧ, пытается использовать теорию сложности так, чтобы решение задачи криптоанализа было бы эквивалентно решению трудной теоретической задачи.

Эди Шамир предложил использовать алгоритм шифрования с открытым ключом RSA для генерации ПСЧ [19]. В его работе показано, что предсказание выхода генератора псевдослучайных чисел равносильно взлому RSA. Очевидным недостатком такого алгоритма является низкая скорость и громоздкость реализации.

Дальнейшей модификацией этого алгоритма является ГПСЧ RSA, который также основан на сложности решения проблемы RSA.

Описание алгоритма.

1. Сгенерировать два секретных простых числа p и q, а также $n = pq$ и $f = (p-1)(q-1)$. Выбрать случайное целое число $e, 1 < e < f$, такое что НОД$(e, f) = 1$.
2. Выбрать случайное целое x_0 – начальный вектор из интервала $[1, n-1]$.
3. $For\ i = 1\ to\ l\ do$
   a. $x_i \leftarrow x_{i-1}^e\ mod\ n$.
   b. $z_i \leftarrow$ последний значащий бит x_i
4. Вернуть $z_1, z_2, \dots, z_l$.

Примечание. Если $e = 3$, генерация одного бита z_i требует одного умножения и одного возведения в квадрат по модулю. Эффективность может быть улучшена, если на шаге 3.а извлекать j последние значащие биты x_i, где $j = c \lg \lg n$, а c – константа.

4.7.3. Модификация алгоритма RSA Микали-Шнорра

Описание алгоритма

1. Сгенерировать два простых p, q и $n = pq$, а также $f = (p-1)(q-1)$. $N = \lfloor \ln n \rfloor + 1$ (длина n). Выбрать целое $e, 1 < e < f$ такое, что НОД$(e, f) = 1$ и $80e \leq N$. Установим $k = \left\lfloor N\left(1 - \frac{2}{e}\right)\right\rfloor$ и $r = N - k$.
2. Выбрать случайный инициализирующий вектор длины r.
3. $For\ i = 1\ to\ l\ do$
   a. $y_i \leftarrow x_{i-1}^e\ mod\ n$.
   b. $x_i \leftarrow$ первые r значимых биты y_i.
   c. $z_i \leftarrow$ последние k значимых бита y_i.
4. Возвратить вектор бит $z_1|z_2|\ldots|z_l$.

Примечание. Этот алгоритм более эффективный, чем алгоритм Шамира, так как генерируется сразу $\lfloor N(1 - 2/e) \rfloor$ битов на одно возведение в степень. Например, если $e = 3$ и $N = 1024$, то $k = 341$ битов на одно возведение в степень. Более того, возведение в степень требует только одного возведения в квадрат 683-битного номера по модулю и одного умножения.

4.7.4. ГПСЧ Блюма-Микали

Этот алгоритм использует сложность вычисления дискретных логарифмов. Приведем описание из [10]. Пусть g – простое число, а p – еще одно простое число. Ключ x_0 начинает процесс:

$$x_{i+1} = g^{x_i}\ mod\ p.$$

Очередной бит последовательности определяется по формуле:

$$X_{i+1} = \begin{cases} 1, & x_{i+1} < (p-1)/2, \\ 0, & x_{i+1} \geq (p-1)/2. \end{cases}$$

Если p достаточно велико, чтобы вычисление дискретных логарифмов по модулю p стало невозможным, то этот генератор безопасен.

4.7.5. Алгоритм Блюма—Блюма—Шуба (BBS)

Алгоритм был предложен в 1986 году Ленор Блюм, Мануэлем Блюмом и Майклом Шубом [20]. В основе алгоритма – использование квадратичных остатков по модулю *n*. На текущее время это один из самых простых и быстрых алгоритмов ГПСЧ, использующих вычислительно сложные задачи.

Описание алгоритма [10].

На входе: Длина *l*.

На выходе: Последовательность псевдослучайных бит $z_1, z_2, \dots, z_l$.

1. Сгенерировать два простых числа *p* и *q*, сравнимых с 3 по модулю 4. Произведение этих чисел – *n*=*pq* является целым числом Блюма. Выберем другое случайное целое число *x*, взаимно простое с *n*.
2. Вычислим $x_0 = x^2 \ mod \ n$, которое будет начальным вектором.
3. $For \ i = 1 \ to \ l \ do$
    1. $x_i \leftarrow x_{i-1}^2 \ mod \ n.$
    2. $z_i \leftarrow$ последний значащий бит x_i
4. Вернуть $z_1, z_2, \dots, z_l$.

Интересным достоинством этого генератора является то, что для получения *i*-го бита b_i при известных *p* и *q* достаточно воспользоваться формулой

$$b_i = x_0^{2^i \ mod \ ((p-1)(q-1))} \ mod \ 2.$$

Это дает преимущества данному ГПСЧ при работе с массивами данных с произвольной точкой доступа (random access data).

Безопасность алгоритма основана на сложности разложения *n* на множители. Этот генератор безопасен как для предсказания следующего бита последовательности, так и для определения предыдущего бита. Это свойство следует из свойств задачи разложения *n* на множители.

Недостатком алгоритма является его низкая скорость, что делает его малоприменимым для использования в потоковых шифрах.

5. Тестирование статистических свойств

Наша основная цель - получить последовательность, которая ведет себя так, как будто она является случайной. Один из важных критериев «случайности» - длина периода последовательности. Это важное свойство, но нам нужны критерии того, что последовательность будет выглядеть достаточно случайной.

Так, например, число Пи, являясь иррациональным, не имеет периода в десятичной записи последовательности цифр, его составляющих:

3.14159265358979323846264338327950…

Известно много формул, позволяющих вычислять это число с нужной степенью приближения, одна из которых выглядит так:

$$\sqrt{6\left(\frac{1}{1^2}+\frac{1}{2^2}+\frac{1}{3^2}+\frac{1}{4^2}+\cdots\right)}.$$

Строгость этой формулы не оставляет сомнений в «неслучайности» этого числа, и определить по серии чисел, что эти числа принадлежат данной последовательности, не составляет труда.

Таким образом, проблема определения случайности не ограничивается критерием длины периода последовательности и требует других беспристрастных критериев. Набор таких критериев может быть предоставлен теоретической статистикой, а также разумными соображениями о том, какие закономерности должны присутствовать в последовательности случайных чисел, а какие нет. Далее будут рассмотрены некоторые самые значимые критерии.

При этом успешная проверка на случайность одним или несколькими критериями не дает нам гарантии качества ПСП. Например, проверив последовательность десятью критериями, она может быть «забракована» одиннадцатым. Тем не менее проверка этими десятью критериями дает нам достаточно большую уверенность в качестве ПСП.

5.1. Виды тестов

Таксономия тестов случайных и псевдослучайных чисел очень обширна. Из самых обобщенных соображений можно выделить два класса критериев: *эмпирические критерии*, при использовании которых вычисляются некоторые статистики от групп чисел ПСП; и *теоретические критерии*, для которых анализ

последовательности чисел производится теоретико-числовыми методами, над рекуррентными правилами, образующими ПСЧ.

Кроме того, тесты ПСЧ можно разделить по способам оценки результата. Общая задача для всех тестов одна – установить проходит ли данная последовательность статистический тест или нет. В результате вычислений над последовательностью мы получаем некоторую статистику, которую можно оценить. А вот оценивать эту статистику можно разными способами:

- сравнение с пороговым значением – последовательность не проходит тест, если значение статистики меньше некоторого порогового значения;
- сравнение с фиксированным диапазоном допустимых значений – последовательность не проходит тест, если выходит за границы некоторого фиксированного диапазона;
- сравнение со значением вероятности статистической величины – последовательность проходит тест, если вероятность вычисленной статистики попадает в ожидаемый диапазон.

Критерии, как правило, применяются к последовательности независимых друг от друга действительных равномерно распределенных чисел из интервала [0,1]: $X^n = (x_1, x_2, \dots, x_n)$. Некоторые из критериев предназначены для целочисленных последовательностей. В этом случае будем использовать вспомогательную последовательность $Y^n = (d\lfloor x_1 \rfloor, d\lfloor x_2 \rfloor, \dots, d\lfloor n \rfloor)$.

В сети Интернет сейчас можно найти много программных макетов, проверяющих ПСЧ. Приведем некоторые из них.

- Программа Statistica компании StatSoft содержит тесты для проверки принадлежности ПСЧ заданному распределению [37];
- Statistics Toolbox\Hypothesis Tests в программе MathLab содержит функции тестирования статистических гипотез [38];
- NIST Statistical Test Suite [31];
- TEST-U01 [32];
- CRYPT-X [33];
- The pLab Project [34];
- DIEHARD [35];
- ENT [36];
- Dieharder [39].

Текстовое описание тестов и алгоритмов дано в книгах [1,3]. Кроме того, существуют рекомендации по тестированию битовых последовательностей NIST SP 800-22 [40]. Обзор всех этих тестов выходит за рамки данного курса. В данной книге дано общее представление о том, на каких принципах строятся такие тесты.

5.2. Критерий хи-квадрат (χ^2-критерий)

Критерий хи-квадрат широко применяется в статистических приложениях для проверки гипотезы о том, что некоторая выборка чисел $X^n = (x_1, x_2, \dots, x_n)$ подчиняется некоторому закону распределения $F(x)$. Критерий имеет другое название – критерий согласия Пирсона. Он является базовым для многих других критериев проверки статистических гипотез.

Описание критерия приведено из [28].

Гипотеза H_0: случайная величина X подчиняется закону ния $F(x)$.

Для проверки гипотезы рассмотрим выборку, состоящую из n независимых наблюдений над случайной величиной X:

$$X^n = (x_1, x_2, \dots, x_n),\ x_i \in [a, b], i = 1,2, \dots, n.$$

По выборке построим эмпирическое распределение $F^*(x)$ случайной величины X. Сравнение эмпирического $F^*(x)$ и теоретического ния $F(x)$ (предполагаемого в гипотезе) производится с помощью специально подобранной функции — критерия согласия. Рассмотрим критерий согласия Пирсона (критерий χ^2):

Гипотеза $\boldsymbol{H_0^*}$: X^n порождается функцией $F^*(x)$.

Разделим $[a, b]$ на k непересекающихся интервалов $(a_i, b_i], i = 1, \dots, k$;

Пусть n_j - количество наблюдений в j-м интервале:

$$n_j = \sum_{i=1}^{n} [a_j < x_j \le b_j];$$

$p_j = F(b_j) - F(a_j)$ – вероятность попадания наблюдения в j-ый интервал при выполнении гипотезы H_0^*;

$E_j = np_j$- ожидаемое число попаданий в j-й интервал;

Статистика:

$$\chi^2 = \sum_{j=1}^{k} \frac{\left(n_j - E_j\right)^2}{E_j} \sim \chi^2_{k-1}$$

называется распределением хи-квадрат с k-1 степенью свободы.

Проверка гипотезы H_0.

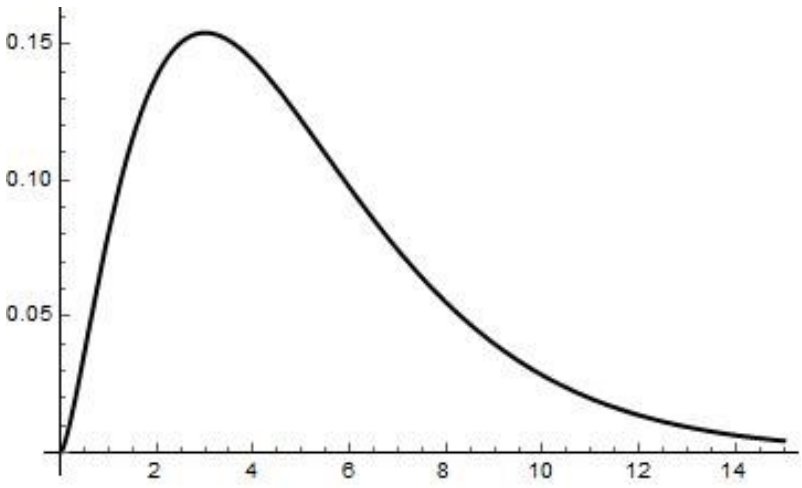

Рисунок 5.1. Распределение хи-квадрат для n=5.

В зависимости от значения критерия χ^2, гипотеза H_0 может приниматься либо отвергаться:

$\chi_1^2 < \chi^2 < \chi_2^2$, гипотеза H_0 выполняется.

$\chi^2 < \chi_1^2$ – попадает в левый «хвост» распределения, означает что теоретические и практические значения очень близки, и гипотеза H_0 принимается.

$\chi^2 \geq \chi_2^2$ – попадает в правый «хвост» распределения, - гипотеза отвергается.

Для точного определения соответствия вычисленной характеристики распределению хи-квадрат используют понятие квантилей распределения хи-квадрат.

Определение 5.1. Квантиль хи-квадрат ($\alpha-$квантиль) – это величина $\chi^2_{\alpha,n}$, при которой функция распределения хи-квадрат равна заданной вероятности α:

$$F_n\left(\chi^2_{\alpha,n}\right) = \alpha,$$

где F_n- функция распределения хи-квадрат с n степенями свободы, и $\alpha \in [0,1]$.

В итоге, проверка критерия χ^2 для некоторой последовательности чисел (или наблюдений величины X) будет состоять из следующих шагов:

1. Выполняем достаточное число независимых наблюдений.
2. Подсчитываем число n_i наблюдений попавших в каждый из интервалов $(a_i, b_i], i = 1, \dots, k$.
3. Подсчитываем статистику

$$\chi^2 = \sum_{j=1}^{k} \frac{\left(n_j - E_j\right)^2}{E_j}.$$

4. Определяем, находится ли вычисленная в доверительном интервале (например, по таблице 5.1).

Для проверки принадлежности заданному распределению случайной величины критерием хи-квадрат, в качестве доверительного интервала можно использовать $[0{,}05; 0{,}95]$.

Если вычисленная величина χ^2 находится в интервале [0; 0,1), – это выглядит неправдоподобно – распределение случайной величины подтверждается наблюдениями слишком точно, что в природе бывает редко (подкинуть 100 раз монету и получить ровно 50 на 50 не легче, чем 95 на 5). Если же величина находится в интервале $(0{,}95; 1]$ – это означает, что гипотеза принадлежности значений наблюдаемой величины не подтвердилась.

На практике часто не вычисляют α, а используют таблицу α-квантилей, строки которой соответствуют числу степеней свободы, а колонки соответствуют значениям α. Здесь приведена часть такой таблицы.

Таблица 5.1. Таблица квантилей (неполная).

$n\backslash\alpha$	**0,01**	**0,025**	**0,05**	**0,1**	**0,2**	**0,3**	**…**	**0,8**	**0,9**	**0,95**	**0,975**	**0,99**
1	0,0002	0,0010	0,0039	0,0158	0,0642	0,1485	…	1,6424	2,7055	3,8415	5,0239	6,6349
2	0,0201	0,0506	0,1026	0,2107	0,4463	0,7133	…	3,2189	4,6052	5,9915	7,3778	9,2103
3	0,1148	0,2158	0,3518	0,5844	1,0052	1,4237	…	4,6416	6,2514	7,8147	9,3484	11,3449
4	0,2971	0,4844	0,7107	1,0636	1,6488	2,1947	…	5,9886	7,7794	9,4877	11,1433	13,2767
5	0,5543	0,8312	1,1455	1,6103	2,3425	2,9999	…	7,2893	9,2364	11,0705	12,8325	15,0863
6	0,8721	1,2373	1,6354	2,2041	3,0701	3,8276	…	8,5581	10,6446	12,5916	14,4494	16,8119
7	1,2390	1,6899	2,1673	2,8331	3,8223	4,6713	…	9,8032	12,0170	14,0671	16,0128	18,4753
8	1,6465	2,1797	2,7326	3,4895	4,5936	5,5274	…	11,0301	13,3616	15,5073	17,5345	20,0902
9	2,0879	2,7004	3,3251	4,1682	5,3801	6,3933	…	12,2421	14,6837	16,9190	19,0228	21,6660

10	2,5582 3,2470 3,9403 4,8652 6,1791 7,2672 … 13,4420 15,9872 18,3070 20,4832 23,2093

Пример. Проверим гипотезу H_0: если подкинуть монету 100 раз и зафиксировать сторону, которой она падает, то количество падений орлом будет примерно равно количеству падений решкой, то есть, ожидаемые величины 50 и 50. Пусть орлом монета упала 59 раз, а решкой – 41.

У нас всего два варианта – орел и решка, поэтому число степеней свободы $k = 2 - 1 = 1$. Посчитаем статистику

$$\chi^2 = \sum_{j=1}^{k} \frac{\left(n_j - E_j\right)^2}{E_j} = \frac{(41-50)^2}{50} + \frac{(59-50)^2}{50} = 3{,}24.$$

Осталось проверить по таблице квантилей, какому значению α соответствует полученный результат. Значение 3,24 лежит в диапазоне между 0,9 и 0.95, что означает – гипотеза подтвердилась.

5.3. Критерий Колмогорова-Смирнова

Критерий Колмогорова-Смирнова (*КС-критерий*) применяется для проверки гипотезы H_0, согласно которой независимые случайные величины $x_1, x_2, \dots, x_n$ имеют заданную непрерывную функцию распределения $F(x)$.

Описание критерия [30].

Гипотеза H_0: случайная величина X подчиняется закону ния $F(x) \in \mathrm{C}^1(X)$.

Для проверки гипотезы рассмотрим выборку, состоящую из n независимых наблюдений над случайной величиной X:

$X^n = (x_1, x_2, \dots, x_n),\ x_i \in [a, b], i = 1,2, \dots, n.$

$F^*(x)$ – эмпирическая функция распределения, $F(x)$ – некоторая «истинная» функция распределения с известными параметрами.

Статистика КС-критерия:

$$D_n = \sup_{|x|<\infty} |F^*(x) - F(x)| = \max_{1 \le m \le n} \left(\frac{m}{n} - F\left(X_{(m)}\right) \right),$$

где $X_{(1)} \le X_{(2)} \le \cdots \le X_{(n)}$ – вариационный ряд, полученный по выборке X^n, который в дальнейшем мы будем обозначать как исходную выборку $x_1, x_2, \dots, x_n$.

Тогда по теореме Колмогорова при справедливости проверяемой гипотезы:

$$\forall t > 0: \lim_{n\to\infty} P\left(\sqrt{n}D_n \le t\right) = K(t) = \sum_{j=-\infty}^{+\infty} (-1)^j e^{-2j^2 t^2}.$$

Гипотеза H_0 отвергается, если статистика $\sqrt{n}D_n$ превышает квантиль распределения K_α заданного уровня значимости α, и принимается в противном случае.

Если α достаточно близко к 1, то K_α можно приблизительно рассчитать по формуле:

$$K_\alpha \approx \sqrt{-\frac{1}{2}\ln\frac{1-\alpha}{2}}.$$

При проверке гипотез с применением критерия Колмогорова рекомендуется использовать статистику с поправкой Большева [30] в форме

$$t_K = \frac{6nD_n + 1}{6\sqrt{n}},$$

где $D_n = \max(D_n^+, D_n^-)$,

$D_n^+ = \max_{1\le i\le n}\left\{\frac{i}{n} - F(x_i)\right\},\ D_n^- = \max_{1\le i\le n}\left\{F(x_i) - \frac{i-1}{n}\right\},\ x_1 \le x_2 \le \cdots \le x_n.$

Здесь D_n^+ определяет наибольшее из отклонений, когда F^* больше F, а D_n^- - F^* меньше F.

Точно так же, как и для χ^2-критерия, можно сравнить значения D_n^+ и D_n^- с таблицей квантилей и определить, будут ли они значимо выше или ниже.

Таким образом, можно предположить следующий алгоритм проверки гипотезы.

1. Получить независимые наблюдения $x_1, x_2, \ldots, x_n$.
2. Упорядочить наблюдения так, чтобы они располагались в порядке возрастания: $x_1 \le x_2 \le \cdots \le x_n$.
3. Вычислить статистики:

$$D_n^+ = \max_{1\le i\le n}\left\{\frac{i}{n} - F(x_i)\right\},\ D_n^- = \max_{1\le i\le n}\left\{F(x_i) - \frac{i-1}{n}\right\}.$$

4. Сравниваем статистики с доверительным интервалом (например, по таблице квантилей критерия Колмогорова).

5.4. Критерий равномерности

Для проверки равномерности распределения последовательности чисел можно воспользоваться критерием Колмогорова-Смирнова с $F(x) = x$ для $x \in [0,1]$. Другой вариант – воспользоваться критерием хи-квадрат [1]. Для этого преобразуем последовательность $X^n = (x_1, x_2, \dots, x_n)$ в $Y^n = (d\lfloor x_1 \rfloor, d\lfloor x_2 \rfloor, \dots, d\lfloor n \rfloor)$ с некоторым d (например, 64). После этого подсчитаем количество элементов Y^n равных n_j, для $j = 0,1, \dots, d-1$ и применим критерий хи-квадрат, принимая

$$k = d, p_j = \frac{1}{d}.$$

5.5. Критерий серий

Описание этого и некоторых следующих критериев взято из [1]. Критерий серий позволяет убедиться в том, что пары последовательных чисел равномерно распределены независимым образом. Проверка критерия проводится по аналогии с предыдущим случаем, однако, считать будем количество совпадений

$$\left(y_{2j}, y_{2j+1}\right) = (q, r), \qquad 0 \le j \le n, \;\; 0 \le q, r \le d.$$

Хи-квадрат критерий применяем к полученному набору с параметрами

$$k = d^2, p_j = \frac{1}{d^2}.$$

Примечание. Для достоверности результатов, полученных критерием хи-квадрат, величина проверяемой выборки должна быть значительно больше, чем количество возможных интервалов (значений). Учитывая это, рекомендуется выбирать длину выборки, по крайней мере, $n \ge 5d^2$ или уменьшать значение d.

Критерий можно обобщить на тройки, четверки и т.д. случайных величин. Однако ширина проверяемого диапазона увеличивается пропорционально степени, равной количеству величин в кортеже. Поэтому на практике, при рассмотрении больших серий чисел, используются менее точные критерии.

5.6. Критерий интервалов

Пусть a и b – два действительных числа таких, что $0 \le a < b \le 1$. Рассмотрим длины подпоследовательностей $x_j, x_{j+1}, \dots, x_{j+r}$, в которых

$x_j, x_{j+1}, \dots, x_{j+r-1} \notin [a, b],\ x_{j+r} \in [a, b]$. Такую последовательность будем называть интервалом длины r.

Сначала, нам нужно подсчитать число интервалов длиной $0,1, \dots, n$.

Шаги алгоритма подсчета числа интервалов [1]:

1. Инициализация. Присвоить $j = -1, s = 0, c_r = 0,\ 0 \le r \le t$.
2. $r = 0$.
3. $j = j + 1$. Если $a \le x_j \le b$, то переход на шаг 5.
4. $r = r + 1$. Переход к шагу 3.
5. Если $r \ge t$, то $c_t = c_t + 1$, иначе – $c_r = c_r + 1$.
6. $s = s + 1$. Если $s < n$ то переход на шаг 2.

После этого мы можем применить хи-квадрат критерий для $k = t + 1$ к значениям $c_i,\ i = 0,1, \dots, t$ с параметрами

$$p_r = p(1-p)^r \text{ для } 0 \le r \le t-1;$$
$$p_t = (1-p)^t;$$
$$p = (a-b).$$

Здесь p – вероятность того, что $a \le x_j \le b$. Значения *n* и *t* выбираются так, чтобы ожидаемое значение c_r было больше 5. Критерий часто применяют для $[a, b] = [0,1]$. В этом случае на шаге 3 алгоритма можно обойтись без сравнения.

Частным случаем применения критерия интервалов является проверка с параметрами

$$(a, b) = \left(0, \frac{1}{2}\right) \text{ и } (a, b) = \left(\frac{1}{2}, 1\right).$$

Эти случаи называют проверкой «отклонения выше среднего» и проверкой «отклонения ниже среднего».

5.7. Критерий разбиений

В общем случае критерия разбиений рассматриваются *n* групп *k* последовательных чисел, и подсчитывается число групп из *k* чисел с *r* различными чис-

лами. Затем применяется хи-квадрат критерий, в котором используются вероятности того, что в группе *r* различных чисел

$$p_r = \frac{d(d-1)\dots(d-r+1)}{d^k}\left\{{k \atop r}\right\}.$$

Здесь $\left\{{k \atop r}\right\} = S(n,k)$ – числа Стирлинга, задающие число способов разбиения множества из n элементов на k непересекающихся подмножеств, которые можно вычислить по формуле:

$$S(n,k) = \frac{1}{k!}\sum_{j=0}^{k}(-1)^{k+j}\binom{k}{j} j^n,$$

$$\binom{k}{j} = \frac{k!}{j!\,(k-j)!}.$$

Так как вероятности p_r очень малы, когда $r = 1$ или 2, следует, перед применением критерия хи-квадрат, объединить несколько категорий, имеющих малые вероятности в одну.

Чтобы получить формулу для p_r, следует подсчитать, сколько d^k групп из k чисел, расположенных между 0 и $d-1$, имеют точно r различных элементов, и разделить это число на d^k.

5.8. Критерий перестановок

Последовательность $X^m = (x_1, x_2, \dots, x_m)$ разбивается на n групп по t элементов в каждой:

$$u_j = \left(x_{jt}, x_{jt+1}, \dots, x_{jt+t-1}\right), \qquad 0 \le j < n.$$

Элементы в каждой группе можно упорядочивать $t!$ различными способами. Подсчитывается число групп с любым возможным порядком и применяется хи-квадрат критерий с $k = t!$ возможными категориями и вероятностью $1/t!$ для каждой категории. Например, для $t = 2$, существует две категории: $x_{2j} > x_{2j+1}$ или $x_{2j+1} > x_{2j}$. Для $t = 3$ таких категорий будет уже шесть: $x_{3j} < x_{3j+1} < x_{3j+2}$ или , $x_{3j} < x_{3j+2} < x_{3j+1}$, или $x_{3j+1} < x_{3j} < x_{3j+2}$, и т.д., …, или $x_{3j+2} < x_{3j+1} < x_{3j}$. В этом критерии предполагается, что x_s не могут быть равны между собой.

5.9. Критерий монотонности

Последовательность можно проверить на предмет равномерности распределения монотонных серий чисел.

5.9.1. Первый вариант критерия монотонности

Суть метода в том, чтобы проверить длины всех восходящих (нисходящих) серий c_i в последовательности и подсчитать для них следующую статистику

$$M = \frac{1}{n-6} \sum_{1 \le i,j \le 6} (c_i - nb_i)(c_j - nb_j) a_{ij},$$

где n – длина последовательности, а матрицы коэффициентов $A = (a_{ij})$, $B = (b_i)$ следующие:

$$A = \begin{pmatrix} 4529.4 & 9044.9 & 13568 & 22615 & 22615 & 27892 \\ 9044.9 & 18097 & 27139 & 36187 & 45234 & 55789 \\ 13568 & 27139 & 40721 & 54281 & 67582 & 83685 \\ 18091 & 36187 & 54281 & 72414 & 90470 & 111580 \\ 22615 & 45234 & 67852 & 90470 & 113262 & 139476 \\ 27892 & 55789 & 83685 & 111580 & 139476 & 172860 \end{pmatrix}, \quad B = \begin{pmatrix} \frac{1}{6} \\ \frac{5}{24} \\ \frac{11}{120} \\ \frac{19}{720} \\ \frac{29}{5040} \\ \frac{1}{840} \end{pmatrix}.$$

Далее, к полученной статистике применяем критерий хи-квадрат с шестью степенями свободы, когда n – большое (например, больше 4000).

Примечание. В данном случае нельзя использовать к длинам серий критерий хи-квадрат, так как проявляется закономерность чередования длинных серий с короткими: вероятность того, что после длинной последовательности будет короткая, выше, чем вероятность длинной серии после длинной. А по условиям критерия хи-квадрат значения проверяемых величин должны быть независимы.

Примечание. Значения элементов матрицы A приведены приблизительно. Правила расчета точных значений приведены в [1, §3.3.2].

5.9.2. Второй вариант критерия монотонности

Для решения проблемы чередования длинных серий с короткими сериями можно сделать следующее.

1. «Выбрасываем» элемент последовательности, который следует непосредственно за серией.
2. Если x_j больше x_{j+1}, то начнем следующую серию с x_{j+2}.
3. Мы получаем серии, длины которых независимы и, поэтому, можно использовать критерий хи-квадрат.

Такой вариант алгоритма проверки критерия монотонности гораздо проще в реализации, чем описанный выше.

5.10. Критерий конфликтов

Предположим, нам нужно оценить последовательность случайных чисел, в которой число величин в последовательности намного меньше числа категорий. В этом случае критерий хи-квадрат не применим, но можно использовать критерий конфликтов [1].

Предположим, что у нас m урн и n шаров, причем m значительно больше n. Если разместить шары в урнах наугад, то некоторые урны останутся пустыми, а в некоторых будет более одного шара. Когда в одну урну попадает больше одного шара, то говорят, что произошел «конфликт». Критерий конфликтов состоит в подсчете и оценке количества конфликтов.

Рассмотрим пример, когда $m = 2^{20}$, а $n = 2^{14}$. В среднем, число урн, приходящихся на один шар – 64. Вероятность того, что в конкретную урну попадет ровно k шаров, равна

$$p_k = \binom{n}{k} m^{-k}(1 - m^{-1})^{n-k},$$

отсюда, среднее число конфликтов в урне вычисляется по формуле

$$\sum_{k\geq 1}(k-1)p_k = \sum_{k\geq 1} kp_k - \sum_{k\geq 1} p_k = \frac{n}{m} - 1 + p_0.$$

Так как $p_0 = (1 - m^{-1})^n = 1 - nm^{-1} + \binom{n}{2} m^{-2}$ – маленькое число, получим, что общее среднее число конфликтов во всех m урнах намного меньше

$$\frac{n^2}{2m} = 128.$$

Этот критерий хорош, когда ГПСЧ выдает строки большой размерности.

5.11. Критерий промежутков между днями рождений

Критерий был введен Дж. Марсальей в 1984 году. Опишем критерий согласно [1, п. 3.3.2]. Предположим, что $(Y_1, Y_2, \dots, Y_m)$ – это дни рождения, где $0 \le Y_k < m$. Расположим их в порядке неубывания $Y_1 \le \cdots \le Y_n$, определим n "промежутков"

$$S_1 = Y_2 - Y_1, \dots, S_{n-1} = Y_n - Y_{n-1}, S_n = Y_1 + m - Y_n$$

и, наконец, расположим промежутки в таком порядке: $S_1 \le \cdots \le S_n$. Пусть R – число равных промежутков, а именно – число индексов j, таких, что $1 < j \le n$ и $S_j = S_{j-1}$. Когда $m = 2^{25}$ и $n = 512$, должно получиться следующее.

R	0	1	2	3 и больше
С вероятностью	0.368801577	0.369035243	0.183471182	0.078691997

В среднем, число одинаковых промежутков для выбранных m и n должно быть равным приблизительно 1. Далее, мы можем применить этот критерий много раз и воспользоваться χ^2-критерием с тремя степенями свободы, чтобы сравнить эмпирические значения R_i с правильным распределением. Так можно узнать, будет ли генератор вырабатывать приемлемые случайные промежутки между днями рождений.

Данный критерий примечателен тем, что на нём споткнулся генератор Фибоначчи с запаздыванием и его производные. Рассмотрим, например, последовательность

$$x_n = (x_{n-24} + x_{n-55}) \bmod m$$

из п. 3.3. Числа этой последовательности удовлетворяют соотношению

$$(x_n + x_{n-86}) \bmod m \equiv (x_{n-24} + x_{n-31}) \bmod m,$$

так как обе стороны конгруэнтны $x_{n-24} + x_{n-55} + x_{n-86}$. Поэтому две пары разностей равны

$$x_n - x_{n-24} \equiv x_{n-31} - x_{n-86}$$

и

$$x_n - x_{n-31} \equiv x_{n-24} - x_{n-86}.$$

Т. о., если x_n оказывается близко к x_{n-24} или x_{n-31}, одна и та же разность имеет больше вероятности появиться в двух промежутках. Тогда в среднем R будет приближаться к 2, а не к 1.

5.12. Универсальный статистический тест Маурера

Универсальный статистический тест Маурера основан на идее, что не возможно существенно сжать последовательность битов, если они действительно случайны. Другими словами, если получилось значительно сжать последовательность бит s, сгенерированную ГПСЧ, то ГПСЧ имеет дефект. Вместо сжатия данных, в этом тесте вычисляется значение, зависящее от длины сжатой последовательности.

Универсальность теста Маурера в том, что он способен обнаружить любой из основных классов дефектов ГПСБ.

Сначала выбираются параметры тестирования: параметр L – длина блока в битах. Последовательность s разбивается на последовательные непересекающиеся блоки длины L и вычисляем из них новую последовательность целых чисел b_i, чье двоичное представление – это i-й блок. Далее определим таблицу T^k

$$T^k = (t_1, t_2, \dots, t_k),$$

где $k = \lfloor N/L \rfloor$ и $t_j \in \{0,1,2, \dots, 2^L - 1\}$. Оставшиеся биты в конце последовательности отбрасываются.

Первые Q блоков называют инициализирующей частью. Q должно быть выбрано как минимум порядка $10 \cdot 2^L$, с тем расчётом, чтобы каждый из 2^L L-битовых блоков появляется как минимум раз в первых Q блоках. Оставшиеся K блоков используются для вычисления статистики X_u следующим образом. Для каждого I, $Q + 1 \le i \le Q + K, A_i = i - T_{b_i}$; A_i – это число позиций, начиная с последнего появления блока b_i.

Тогда

$$X_u = \frac{1}{K} \sum_{i=Q+1}^{Q+K} \log_2 A_i.$$

Величина X_u аппроксимируется с нормальным распределением. *K* должно быть как минимум $1000 \cdot 2^L$ (и, следовательно, последовательность *s* должна состоять не менее чем из $1010 \cdot 2^L \cdot L$ битов.

Описание алгоритма.

На входе: $s^n = (s_1, s_2, \dots, s_N)$ – бинарная последовательность длины *N*. Параметры *L, Q, K*.

На выходе: Значение статистики X_u для последовательности s.

1. $For\ j = 0\ to\ 2^L - 1\ do: T[j] \leftarrow 0.$
2. $For\ i = 1\ to\ Q\ do: T[b_i] \leftarrow i.$
3. $sum \leftarrow 0.$
4. $For\ i = Q + 1\ to\ Q + K\ do$
    a. $sum \leftarrow sum + \lg(i - T[b_i]).$
    b. $T[b_i] \leftarrow i.$
5. $X_u \leftarrow \frac{sum}{K}.$
6. Вернуть X_u.

Универсальный статистический тест Маурера использует вычисленное значение X_u для последовательности *s* в соответствии с утверждением:

Утверждение 5.1. Пусть X_u статистика, определенная выше, имеет значения μ и σ^2 приведенные в таблице 5.1. Тогда, для случайной последовательности, статистика $Z_u = (X_u - \mu)/\sigma$ приблизительно соответствует распределению $N(0,1)$.

5.13. Статистические критерии NIST

Отдельно рассмотрим Статистические критерии NIST (Statistical Testing Suite of Random Number Generators или NIST STS) – программный пакет, разработанный Лабораторией информационных технологий (ITL NIST). Всего в составе пакета 15 тестов, предназначенных для проверки бинарных последовательностей, сгенерированных ГПСЧ. Каждый из тестов вычисляет статистику, позволяющую выявить один из дефектов, присущих неслучайным последовательностям. Базовыми идеями этих тестов являются тесты, описанные Д. Кнутом в своей книге [1] (основные приведены выше в п. 5.2-5.11), а также тесты, разработанные Дж. Марсальей в пакете DIEHARD [35]. Однако все тесты в пакете приведены к единой вычислительной схеме, состоящей из четырех шагов:

1. Основной проверяемой гипотезой является случайность последовательности бит. Её мы и будем проверять.
2. Вычисляем статистику для двоичной последовательности.
3. Вычисляем p-значение из интервала [0,1].
4. Сравниваем p-значение с $\alpha \in (0{,}001, 0.01]$. Если p-значение не превышает α, то считаем, что последовательность прошла тест. Иначе – нет.

Рассмотрим кратко основные тесты NIST STS.

5.13.1. Частотный побитовый тест (The Frequency Monobit Test)

Цель теста – выявить дисбаланс в количестве нулей и единиц в последовательности. Тест считается проваленным, если в последовательности слишком много нулей или слишком много единиц.

Рекомендуемой длиной тестируемых последовательностей согласно NIST STRN являются битовые строки по 1 000 000 бит. Всего тестируется для принятия одной из гипотез 100 последовательностей.

5.13.2. Тест кумулятивных сумм (The Cumulative Sums Test)

Тест заключается в максимальном отклонении (от нуля) при произвольном обходе, определяемым кумулятивной суммой заданных (-1, +1) цифр в последовательности. Цель данного теста — определить является ли кумулятивная сумма частичных последовательностей, возникающих во входной последовательности, слишком большой или слишком маленькой по сравнению с ожидаемым поведением такой суммы для абсолютно произвольной входной последовательности. Таким образом, кумулятивная сумма может рассматриваться как произвольный обход. Для случайной последовательности отклонение от произвольного обхода должно быть вблизи нуля.

5.13.3. Блочный тест на частоту (Test for Frequency within a Block)

Последовательность разбивается на блоки длиной M бит, и для каждого рассчитывается частота появления единиц и насколько она близка к эталонному значению – M/2. Когда M=1, длина блока 1 бит и тест равнозначен предыдущему. Длина тестовой последовательности не менее 100 бит, длина блока больше 20 бит.

5.13.4. Тест на серийность (Runs Test)

В тесте находятся все серии битов – непрерывные последовательности одинаковых битов – и их распределение сравнивается с ожидаемым распределением таких серий для случайной последовательности. Длина последовательности 100 и более бит.

5.13.5. Матрично-ранговый тест (Random Binary Matrix Rank Test)

Цель теста – проверка линейной зависимости между подстроками фиксированного размера – матрицами 32x32 бита. Для этого осуществляется расчет рангов непересекающихся подматриц, построенных из исходной двоичной последовательности. Длина последовательности – не менее 38 912 бит, или 38 матриц.

5.13.6. Спектральный тест (тест дискретным преобразованием Фурье)

Цель теста – обнаружить повторяющиеся блоки или последовательности. Для этого последовательность преобразуется по формуле преобразования Фурье, после чего оцениваются высоты пиков дискретного преобразования Фурье исходной последовательности.

5.13.7. Тест с неперекрывающимися непериодическими шаблонами (Non-overlapping Template Matching Test)

Показывает число заранее заданных битовых строк (шаблонов) в последовательности. Поиск шаблонов осуществляется методом бегущего окна, с подсчетом найденных совпадений. В этом тесте, при найденном совпадении окно перескакивает на следующий за обнаруженным шаблоном бит.

5.13.8. Тест на перекрывающиеся периодические шаблоны (Overlapping (Periodic) Template Matching Test)

Показывает количество заранее определенных шаблонов (периодичных битовых последовательностей) в тестовой последовательности. Аналогично предыдущему тесту, шаблоны ищутся методом бегущего окна, но в случае обнаружения шаблона, сдвиг производится как обычно, на один бит.

5.13.9. Универсальный статистический тест (Maurer's Universal Statistical Test)

Показывает число бит между двумя шаблонами и служит для определения сжимаемости последовательности. Подробно он рассмотрен в п. 4.12.

5.13.10. Тест приблизительной энтропии (The Approximate Entropy Test)

Как и в тесте на периодичность в данном тесте акцент делается на подсчете частоты всех возможных перекрытий шаблонов длины m бит на протяжении исходной последовательности битов. Цель теста — сравнить частоты перекрывания двух последовательных блоков исходной последовательности с длинами m и m+1 с частотами перекрывания аналогичных блоков в абсолютно случайной последовательности.

5.13.11. Тест на произвольные отклонения (The Random Excursions Test)

Суть данного теста заключается в подсчете числа циклов, имеющих строго k посещений при произвольном обходе кумулятивной суммы. Произвольный обход кумулятивной суммы начинается с частичных сумм после последовательности (0,1), переведенной в соответствующую последовательность (-1, +1). Цикл произвольного обхода состоит из серии шагов единичной длины, совершаемых в произвольном порядке. Кроме того, такой обход начинается и заканчивается на одном и том же элементе. Цель данного теста — определить отличается ли число посещений определенного состояния внутри цикла от аналогичного числа в случае абсолютно случайной входной последовательности. Фактически данный тест есть набор, состоящий из восьми тестов, проводимых для каждого из восьми состояний цикла: −4, −3, −2, −1 и +1, +2, +3, +4.

5.13.12. Другой тест на произвольные отклонения (*The Random Excursions Variant Test*)

В этом тесте подсчитывается общее число посещений определенного состояния при произвольном обходе кумулятивной суммы. Целью является определение отклонений от ожидаемого числа посещений различных состояний при произвольном обходе. В действительности этот тест состоит из 18 тестов, про-

водимых для каждого состояния: −9, −8, …, −1 и +1, +2, …, +9. На каждом этапе делается вывод о случайности входной последовательности.

5.13.13. Тест на периодичность (The Serial Test)

Определяет, действительно ли количество появлений 2^m перекрывающихся шаблонов длиной m бит приблизительно такое же, как в случае абсолютно случайной входной последовательности бит. Тест заключается в подсчете частоты всех возможных перекрываний шаблонов длины m бит на протяжении исходной последовательности битов. Предполагается, что в абсолютно случайной последовательности каждый шаблон длиной m бит появляется в последовательности с одинаковой вероятностью.

5.13.14. Комплексный тест Lempel-Ziv (Lempel-Ziv Complexity Test)

Цель теста – определить число четных слов в последовательности и таким образом определить сжимаемость последовательности.

5.13.15. Тест на линейную сложность (Linear Feedback Shift Register)

Оценивает линейную сложность последовательности, воспринимая её как LFSR-последовательность. Абсолютно случайные последовательности характеризуются длинными линейными регистрами сдвига с обратной связью. Если же такой регистр слишком короткий, то предполагается, что последовательность не является в полной мере случайной.

6. Преобразование ПСЧ к нужному распределению

Рассмотрим способы генерации случайных чисел Y, соответствующих различным распределениям.

Будем использовать следующие обозначения:

$F(y)$ – функция распределения,

$f(y)$ – функция плотности вероятности непрерывного распределения,

$p(y)$ – функция плотности дискретного распределения.

Определение 6.1. Случайные числа, соответствующие стандартному равномерному распределению, называют стандартными равномерными случайными числами и обозначают U_1, U_2, U_3, и т.д. Эти числа считают независимыми друг от друга. Мы будем называть их базовыми случайными числами.

Есть два основных класса алгоритмов преобразования базовых случайных чисел U_i в случайные числа Y_i, распределенные по заданному закону распределения.

Первый класс алгоритмов – алгоритмы, использующие метод *инверсии*, в котором базовое число U_i с помощью арифметических операций преобразуется в число Y_i.

Второй класс алгоритмов состоит в моделировании условий соответствующей предельной теоремы теории вероятностей.

Кроме этих двух подходов, можно выделить эвристические способы генерирования случайных чисел заданного распределения.

6.1. Стандартное равномерное распределение

6.1.1. Функция плотности вероятности

$$f(y) = \begin{cases} 1, & \text{если } 0 \le y \le 1, \\ 0, & \text{если } y \notin [0,1]. \end{cases}$$

6.1.2. Метод генерации случайной величины

Если максимальное значение равномерного целого случайного числа X равно $(m-1)$, для генерации стандартных равномерных случайных чисел необходимо применять следующую формулу: $U = X/m$.

Примечание. Поскольку X принимает дискретные значения, величина U также является дискретной.

Примечание. Величина U принимает 0,0 только в том случае, если $X = 0$. Значение 1 величина не принимает.

6.2. Общий случай равномерного распределения

6.2.1. Функция плотности вероятности

$$f(y) = \begin{cases} \frac{1}{b}, & \text{если } a \le y \le a + b, \\ 0, & \text{если } y \notin [a, a + b]. \end{cases}$$

где $b > 0$.

6.2.2. Метод генерации случайной величины

Если стандартное равномерное случайное число U получено методом, установленным в предыдущем параграфе, то равномерное случайное число должно быть получено в соответствии со следующей формулой

$$Y = bU + a.$$

6.3. Треугольное распределение

6.3.1. Функция плотности вероятности

$$f(y) = \begin{cases} \frac{b - |a - y|}{b^2}, & \text{если } a - b \le y \le a + b, \\ 0, & \text{если } y \notin [a - b, a + b]. \end{cases}$$

где $b > 0$.

6.3.2. Метод генерации случайной величины

Если стандартные случайные числа U_1 и U_2 независимо получены методом генерации стандартного равномерного числа, то случайное число Y, подчиняющееся треугольному распределению, определяют по формуле $Y = a + b(U_1 + U_2 - 1)$.

6.4. Общее экспоненциальное распределение с параметрами положения и масштаба

6.4.1. Функция плотности вероятности

$$f(y) = \begin{cases} \frac{1}{b}\exp\left\{-\frac{y-a}{b}\right\}, & \text{если } y \geq a, \\ 0, & \text{если } y < a. \end{cases}$$

где a и b – параметры положения и масштаба экспоненциального распределения соответственно.

6.4.2. Метод генерации случайной величины

Если стандартное равномерное случайное число U генерировано одним из методов, установленным в разделе 2, то случайное число, соответствующее экспоненциальному распределению, получают по формуле

$$Y = -b\ln(U) + a.$$

6.5. Нормальное распределение (распределение Гаусса)

6.5.1. Функция плотности вероятности

$$f(z) = \frac{1}{\sqrt{2\pi}\sigma}\exp\left\{-\frac{1}{2\sigma^2}(z-\mu)^2\right\}, -\infty < z < +\infty,$$

где μ и σ – среднее и стандартное отклонение нормального распределения соответственно.

6.5.2. Метод Бокса-Мюллера (Джордж Бокс, Мервин Мюллер 1958 г.)

Если стандартные равномерные случайные числа U_1 и U_2 независимо сгенерированы методом, установленным в разделе 2, то два независимых нормальных случайных числа Z_1, Z_2 получают в соответствии со следующей процедурой

$$Z_1 = \mu + \sigma\sqrt{-2\ln(1-U_1)}\cos(2\pi U_2),$$

$$Z_2 = \mu + \sigma\sqrt{-2\ln(1-U_1)}\sin(2\pi U_2).$$

Примечание. При получении U_1, U_2 линейным конгруэнтным методом последовательно, U_1 и U_2 являются зависимыми, таким образом хвост распределений, полученных Z_1 и Z_2 может существенно отличаться от истинно нормального распределения.

6.6. Гамма-распределение

6.6.1. Функция гамма-распределения

$$f(y) = \begin{cases} \dfrac{1}{b\Gamma(c)}\left\{\dfrac{y-a}{b}\right\}^{c-1} \exp\left\{-\dfrac{y-a}{b}\right\}, & \text{если } y \geq a, \\ 0, & \text{если } y < a. \end{cases}$$

где *a,b,c* – параметры положения, масштаба и формы соответственно.

6.6.2. Методы генерации случайной величины

Алгоритмы приведены для трех ситуаций в зависимости от значения параметра формы c.

6.6.2.1. Алгоритм для c=k (k – целое число)

Используя независимые равномерные случайные числа $U_1, U_2, \ldots, U_k$, применяют формулу

$$Y = a - b * \ln\{(1-U_1)(1-U_2)\ldots(1-U_k)\}.$$

Примечание. Этим методом для $a = 0$ и $b = 2$ может быть получено распределение Хи-квадрат с четным числом степеней свободы.

6.6.2.2. Алгоритм для c=k+1/2 (k – целое число)

Используя стандартное нормальное случайное число Z_0 и равномерные случайные числа $U_1, U_2, \ldots, U_k$, применяют формулу

$$Y = a + b\left[\frac{Z_0^2}{2} - ln\{(1-U_1)(1-U_2)\ldots(1-U_k)\}\right].$$

В случае, когда *k*=0, член с логарифмом исчезает.

Примечание. Тем же методом при $a = 0$ и $b = 2$ может быть получено распределение Хи-квадрат с нечетным числом степеней свободы.

Приближенный метод для $c > 1/3$:

a) задают $r = c - 1/3$, $s = \sqrt[3]{r}, t = r - r\ln(r)$, $p = \frac{1}{3\sqrt{s}}$ и $q = -3\sqrt{r}$.

b) Генерируют стандартное нормальное случайное число Z.

c) Если $Z < q$, то переходят к b).
d) Вычисляют $Y = (pZ + s)^3, V = Z^2/2$ и генерируют U.
e) Если $(Y - r)^2/Y - V \leq U$, выполняют $Y \coloneqq a + bY$ (конец).
f) Вычисляют $W = Y - r\ln(r) - r - V$.
g) Если $W \leq U$, то выполняют $Y \coloneqq a + bY$ (конец).
h) Если $W > -\ln(1.0 - U)$, то переходят к b).

Примечание. Метод основан на преобразовании Уилсона-Хилферти, приводящем Хи-квадрат-распределение к приближенному стандартному нормальному распределению. Точность такого приближения зависит от значения параметра c. Идея преобразования состоит в следующем: абсолютная разность между процентной точкой приближенного и точного распределений всегда меньше 0,2.

6.6.2.3. Точный метод генерации Ченга для c>1/2.

a) Задают $q = c - \ln 4$ и $r = c + \sqrt{2c - 1}$.
b) Генерируют стандартные равномерные случайные числа U_1 и U_2.
c) Вычисляют $V = c\ln(U/(1 - U_1)), W = ce^{U_1}, Z = U_1^2 U_2, R = q + rV - W$.
d) Если $R \geq 4{,}5Z - (1 + \ln(4{,}5))$, то вычисляют $Y = a + bW$ (выход).
e) Если $R \geq \ln(Z)$, то вычисляют $Y = a + bW$(выход).
f) Генерируют стандартные равномерные случайные числа U_1 и U_2 и вычисляют $p = \frac{1}{\sqrt{2c-1}}, q = c - \ln(4), r = c + \sqrt{2c - 1}$.

если $q + pr\ln\left(\frac{U_1}{1-U_1}\right) - c\left(\frac{U_1}{1-U_1}\right)^p \geq 4{,}5(U_1^2 U_2) - (1 + \ln(4{,}5))$,

то $Y = a + bc\left(\frac{U_1}{1-U_1}\right)^p$ (выход).

6.7. Распределение Вейбулла-Гнеденко

6.7.1. Функция распределение вероятности

$$F(y) = \begin{cases} 1 - \exp\left\{-\left(\dfrac{y-a}{b}\right)^c\right\}, & \text{если } y \geq a, \\ 0, & \text{если } y < a, \end{cases}$$

где a, b, c – параметры положения, масштаба и формы соответственно.

6.7.2. Метод генерации случайной величины

Если стандартные равномерные случайные числа U генерированы методом, установленным нами выше (Равномерное распределение), то случайные числа, соответствующие распределению Вейбулла, получают по формуле $Y = a - b\{\ln(1-U)\}^{1/c}$.

6.8. Логнормальное распределение

6.8.1. Функция плотности вероятности

$$f(y) = \begin{cases} 1/\left(\sqrt{2\pi}\left\{\dfrac{y-a}{b}\right\}\right)\exp\left\{-\dfrac{1}{2}\left(\dfrac{y-a}{b}\right)^2\right\}, & \text{если } y \geq a, \\ 0, \qquad \text{если } y < a. \end{cases}$$

где a и b – параметры положения и масштаба, соответствующего нормального распределения.

6.8.2. Метод генерации случайной величины

Используя стандартные нормальные случайные числа Z, применяют формулу $Y = a + \exp(b - Z)$ для получения случайных чисел, соответствующих логнормальному распределению.

6.9. Логистическое распределение

6.9.1. Функция вероятности

$$F(y) = \frac{1}{1 + \exp\left\{-\dfrac{y-a}{b}\right\}}, -\infty < y < \infty,$$

где a и b – параметры положения и масштаба соответственно.

6.9.2. Метод генерации случайной величины

Если стандартные равномерные случайные числа U генерированы методом, изложенным выше, то случайные числа, соответствующие логистическому распределению, получают по формуле

$$Y = a + b\ln\left(\frac{U}{1-U}\right).$$

6.10. Многомерное нормальное распределение

Случайные числа $Y_1, Y_2, \dots, Y_n$, соответствующие n-мерному нормальному распределению со средними $\mu_1, \mu_2, \dots, \mu_n$, дисперсиями и ковариациями $\sigma_{ij} (1 \leq i, j \leq n)$, получают, используя взаимно независимые стандартные нормальные случайные числа $Z_1, \dots, Z_n$

$$Y_1 = \mu_1 + a_{11} Z_1,$$

$$Y_2 = \mu_2 + a_{21} Z_1 + a_{22} Z_2,$$

…

$$Y_n = \mu_n + a_{n1} Z_1 + a_{n2} Z_2 + \cdots + a_{nn} Z_n,$$

где $a_{11}, \dots a_{nn}$- константы, вычисляемые до начала генерации, в соответствии с процедурой факторизации Холецкого.

Примечание. $-\sigma_{ij} (1 \leq i, j \leq n)$ ковариации, σ_{ij} – дисперсии Y_i.

a) Для $j = 1, a_{11} = \sqrt{\sigma_{11}}$, $a_{i1} = \frac{\sigma_{i1}}{a_{11}}, (2 \leq i \leq n)$.

b) Для $j = 2, \dots, n$

$$a_{jj} = \left(\sigma_{jj} - \sum_{k=1}^{j-1} a_{jk}^2 \right)^{\frac{1}{2}},$$

$$a_{ij} = \frac{\sigma_{ij} - \sum_{k=1}^{j-1} a_{ik}^2 a_{jk}}{a_{jj}} \quad (j + 1 \leq i \leq n).$$

6.11. Биномиальное распределение

6.11.1. Функция распределения

Если вероятность появления события при каждом испытании равна p, то вероятность того, что это событие произойдет y раз за n испытаний, определяют по формуле

$$p(y) = \binom{n}{y} p^y (1-p)^{n-y}, \qquad y = 0,1, \dots, n,$$

где $0 < p < 1$.

6.11.2. Методы генерации случайной величины

Рассматриваемые в данном разделе методы позволяют получить случайные числа *Y*, соответствующие биномиальному распределению.

6.11.2.1. Прямой метод

Генерируют *n* стандартных равномерных случайных чисел *U*. Искомое число *Y* равно количеству чисел *U* менее *p* из *n* полученных чисел *U*.

6.11.2.2. Метод обратной функции

Вычисляют функцию распределения

$$F(y) = \sum_{k=0}^{y} \binom{n}{k} p^k (1-p)^{n-k}, \quad y = 0,1,\dots,n.$$

Для получения случайного числа *Y* генерируют стандартное равномерное случайное число *U*. Случайное число *Y* является наименьшим значением *y*, для которого $U \le F(y)$.

6.11.2.3. Метод положения

Вычисляют $(n+1)$ параметров $v_1, v_2, \dots, v_n$ и $(n+1)$ параметров $a_0, a_1, \dots, a_n$

a) $v_y = (n+1)p(y), y = 0,1,\dots,n.$
b) Составляют набор индексов G таких, для которых соответствующий параметр *v* удовлетворяет условию $v_y \ge 1$, и набор индексов S, для которых соответствующий параметр *v* удовлетворяет условию $v_y < 1$.
c) Для не пустого набора *S* выполняют операции 1)-4).
    1) Выбирают любой элемент *i* из *G* и любой элемент *j* от *S*.
    2) Устанавливают $a_j = i$ и $v_i = v_i - (1 - v_j)$.
    3) Если $v_i < 1$, удаляют элемент i из G и перемещают его в *S*.
    4) Удаляют элемент *j* из *S*.

Если приведенные выше подготовительные действия выполнены, то двухмерное случайное число *Y* получают, выполняя d)-f).

d) Генерируют стандартное равномерное случайное число *U* и вычисляют $V = (n+1)U$.
e) Вычисляют $k = \lfloor V \rfloor$ и $u = V - k$, где $\lfloor V \rfloor$ – целая часть числа *V*.
f) Если $u \le v_k$, то $Y = k$; в противном случае $Y = a_k$.

Список источников

1. Дональд Кнут. Искусство программирования, том 2. Получисленные алгоритмы = The Art of Computer Programming, vol.2. Seminumerical Algorithms. — 3-е изд. — М.: «Вильямс», 2007. — С. 832. — ISBN 0-201-89684-2
2. The RAND Corporation (Author). A Million Random Digits with 100,000 Normal Deviates Paperback – October 23, 2001.
3. A. Menezes, P. van Oorschot, S. Vanstone. Handbook of Applied Cryptography. — CRC-Press, 1996. — 816 p. — (Discrete Mathematics and Its Applications).
4. Intel® Digital Random Number Digital Random Number Generator Generator(DRNG). Software Implementation Guide. Revision 1.1. August 7, 2012. [Электронный ресурс]. – Режим доступа: https://software.intel.com/sites/default/files/m/d/4/1/d/8/441_Intel_R__DRNG_Software_Implementation_Guide_final_Aug7.pdf. Дата 29.07.2016.
5. Аппаратный генератор случайных чисел ГСЧ-6. [Электронный ресурс]. – Режим доступа: http://tegir.ru/ml/k66.html. Проверено 29.07.2016.
6. ГОСТ Р ИСО 28640-2012. [Электронный ресурс]. – Режим доступа: http://files.stroyinf.ru/cgi-bin/ecat/ecat.fcgi?b=0&i=53898&pr=1. Проверено 29.07.2016.
7. http://www.noisecom.com. [Электронный ресурс]. – Режим доступа: Проверено 29.07.2016.
8. Шнайер Б. 14.1 Алгоритм ГОСТ 28147-89 // Прикладная криптография. Протоколы, алгоритмы, исходные тексты на языке Си = Applied Cryptography. Protocols, Algorithms and Source Code in C. — М.: Триумф, 2002. — С. 373-377.
9. Barker E., Kelsey J. NIST Special Publication 800-90A. Recommendation for Random Number Generation Using Deterministic Random Bit Generators.
10. Шнайер Б. Прикладная криптография. Протоколы, алгоритмы, исходные тексты на языке Си = Applied Cryptography. Protocols, Algorithms and Source Code in C. — М.: Триумф, 2002. — 816 с.
11. Lock-in effect in cascades of clock-controlled shift-registers. In Christoph G. Gunther, editor, Advances in Cryptology—EUROCRYPT 88, volume 330 of Lecture Notes in Computer Science, pages 331–344.

12. G. Mayhew, R. Frazee, and M. Bianco, “Kinetic Protection Device”, Proceedings of the 15th National Computer Security Conference, NIST, 1994, pp. 147-154.
13. Ross J. Anderson. On Fibonacci Keystream Generators [Электронный ресурс]. – Режим доступа: http://www.iacr.org/cryptodb/data/paper.php?pubkey=2963. Заглавие с экрана. Проверено 20.07.2016.
14. Н. Смарт. Криптография. – Москва: Техносфера, 2005. 528 с. ISBN 5-94836-043-1.
15. Recommendation for Block Cipher Modes of Operation. NIST Special Publication 800-38A. Technology Administration U.S. Department of Commerce. 2001 Edition.
16. S. Wolfram, “Random Sequence generation by Cellular Automata”, Advances in Applied Mathematics, v. 7, 1986, pp.123-164.
17. W. Meier and O. Staffelbach, “Fast Correlation Attack on Stream Ciphers”, Journal of Cryptology v I n. 3, 1989, pp.159-176.
18. P.H. Bardell, “Analisis of Cellular Automata Used as Pseudorandom Pattern generators”, Proceedings of 1990 International Test Conference, pp. 762-768.
19. A. Shamir. «On the generation of cryptographically strong pseudorandom sequences». Journal: ACM Transactions on Computer Systems - TOCS , vol. 1, no. 1, pp. 38-44, 1983.
20. Lenore Blum, Manuel Blum, and Michael Shub. «A Simple Unpredictable Pseudo-Random Number Generator», SIAM Journal on Computing, volume 15, pages 364—383, May 1986.
21. C. Borrelli. IEEE 802.3 Cyclic Redundancy Check. [Электронный ресурс]. – Режим доступа: http://www.xilinx.com/support/documentation/application_notes/xapp209.pdf. Проверено 20.08.2015.
22. RFC 1320 - The MD4 Message-Digest Algorithm. [Электронный ресурс]. – Режим доступа: http://www.faqs.org/rfcs/rfc1320.html. Проверено 29.07.2016.
23. RFC 1321 - The MD5 Message-Digest Algorithm. [Электронный ресурс]. – Режим доступа: https://tools.ietf.org/html/rfc1321. Проверено 29.07.2016.

24. Xiaoyun Wang, Dengguo Feng, Xuejia Lai, Hongbo Yu. Collisions for Hash Functions MD4, MD5, HAVAL-128 and RIPEMD. [Электронный ресурс]. – Режим доступа: http://eprint.iacr.org/2004/199. Проверено 29.07.2016.
25. ГОСТ Р 34.11-2012: функция хеширования «Стрибог». [Электронный ресурс]. – Режим доступа: https://www.streebog.net/ru/. Проверено 21.07.2016.
26. ГОСТ Р 34.11-2012. Информационная технология. Криптографическая защита информации. Функция хэширования. [Электронный ресурс]. – Режим доступа: http://protect.gost.ru/document.aspx?control=7&id=180209. Проверено 20.08.2015.
27. O. Kazymyrov, V. Kazymyrova. Algebraic Aspects of the Russian Hash Standard GOST R 34.11-2012. [Электронный ресурс]. – Режим доступа: http://eprint.iacr.org/2013/556.pdf. Проверено 20.08.2015.
28. Р 50.1.033–2001. Рекомендации по стандартизации. Прикладная статистика. Правила проверки согласия опытного распределения с теоретическим. Часть I. Критерии типа хи-квадрат. – М.: Изд-во стандартов. 2002. – 87 с.
29. ГОСТ 28147-89. Системы обработки информации. Защита криптографическая. Алгоритм криптографического преобразования. [Электронный ресурс]. – Режим доступа: http://protect.gost.ru/document.aspx?control=7&id=139177. Проверено 29.07.2016.
30. Большев Л. Н., "Теория вероятностей и ее применения", 1963, т. 8, в. 2, с. 129-55.
31. NIST Statistical Test Suite. [Электронный ресурс]. – Режим доступа: http://csrc.nist.gov/groups/ST/toolkit/rng/documentation_software.html. Проверено 23.08.2015.
32. TEST-U01. [Электронный ресурс]. – Режим доступа: http://www.iro.umontreal.ca/~simardr/testu01/tu01.html. Проверено 23.08.2014.
33. CRYPT-X. [Электронный ресурс]. – Режим доступа: http://www.isi.qut.edu.au/resources/cryptx. Проверено 23.08.2015.
34. The pLab Project. [Электронный ресурс]. – Режим доступа: http://random.mat.sbg.ac.at. Проверено 23.08.2015.

35. George Marsaglia, DIEHARD Statistical Tests. [Электронный ресурс]. – Режим доступа: http://stat.fsu.edu/~geo/diehard.html. Проверено 23.08.2014.
36. ENT. [Электронный ресурс]. – Режим доступа: http://www.fourmilab.ch/random/. Проверено 23.08.2015.
37. Официальный сайт StatSoft Russia. [Электронный ресурс]. – Режим доступа. http://www.statsoft.ru. Проверено 23.08.2014.
38. Официальный сайт программы MathLab. [Электронный ресурс]. – Режим доступа. http://matlab.ru/. Проверено 23.08.2014.
39. Robert G. Brown's General Tools Page. [Электронный ресурс]. – Режим доступа. http://www.phy.duke.edu/~rgb/General/dieharder.php. Проверено 23.08.2014.
40. A. Rukhin, J. Soto, and others. A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications. [Электронный ресурс]. – Режим доступа: http://csrc.nist.gov/publications/nistpubs/800-22-rev1a/SP800-22rev1a.pdf. Проверено 25.08.2014.
41. Andrew Chi-Chih Yao. Theory and applications of trapdoor functions. In Proceedings of the 23rd IEEE Symposium on Foundations of Computer Science, 1982.
42. J.A. Reeds, “Solution of Challenge Cipher”, Cryptologia, v.3, n.2, Apr 1979, pp. 83-95.
43. J.B. Plumstead, “Inferring a Sequence generated by a Linear Congruence”, Proceedings of the 23rd IEEE Symposium on the Foundations of Computer Science, 1982, pp. 153-159.
44. Alex Biryukov, Adi Shamir, "Real Time Cryptanalysis of the Alleged A5/1 on a PC (preliminary draft)". [Электронный ресурс]. – Режим доступа: http://cryptome.org/a51-bsw.htm. Проверено 05.01.2015.
45. Vlastimil Klima. “Tunnels in Hash Functions: MD5 Collisions Within a Minute”. [Электронный ресурс]. – Режим доступа: http://eprint.iacr.org/2006/105. Проверено 06.01.2015.
46. Turing A. Programmers' Handbook for the Manchester Electronic Computer Mark II. — 1952. — C. 25. — 110 с.
47. Eichenauer J., Lehn J. A non-linear congruential pseudo random number generator // Statistische Hefte — Springer Berlin Heidelberg, 1986. — Vol. 27, Iss. 1. — P. 315—326. — ISSN 0932-5026 — doi:10.1007/BF02932576

48. Eichenauer-Herrmannn J., Grothe H., Niederreiter H. et al. On the lattice structure of a nonlinear generator with modulus 2^{α} // J. Comput. Appl. Math. — 1990. — Vol. 31, Iss. 1. — P. 81—85. — ISSN 0377-0427 — doi:10.1016/0377-0427(90)90338-Z
49. Иванов М.А., Чугунков И.В. Теория, применение и оценка качества генераторов псевдослучайных последовательеьностей. – М.: КУДИЦ-ОБРАЗ, 2003. – 240 с. (СКБ – специалисту по компьютерной безопасности).
50. Блейхут Р. Быстрые алгоритмы цифровой обработки сигналов. – М.: Мир. — 1989.
51. Берлекэмп Э. Алгебраическая теория кодирования. = Algebraic Coding Theory. – М.: Мир, 1971.
52. Лидл Р., Нидеррайтер Г. Конечные поля. В 2-х тт. — М.: Мир, 1998. — 430 с.
53. Биркгоф Г., Барти Т. Современная прикладная алгебра. – М.: Мир, 1976.
54. N.Zierler, Brillhart J., On primitive trinomial (mod 2), Inform. Control 13 (1968), 541-554.
55. C.G. Gunter, “Alternating Step Generators Contolled by de Bruijn Sequences”, Advances in Cryptology EUROCRYPT ’87 Proceedings, Springer-Verlag, 1988, pp. 5-14.
56. D. Gollmann, “Kaskadenschaltungen takt gesteuerter Schicberegister als Pseudozu fallszahlengencratoren”, Ph.D. dissertation Universitat Linz, 1983.