Лабораторная работа № 4
Генерация криптографически безопасной псевдослучайной последовательности
Цель работы

Изучить алгоритмы и получить навыки генерации криптографически безопасной псевдослучайной последовательности. Научиться тестировать полученную последовательность на равномерность и случайность.

Введение

Случайные числа и их генераторы являются неотъемлемыми элементами современных криптосистем. Можно привести следующие примеры использования случайных чисел в криптологии [2]:

- сеансовые и другие ключи для симметричных криптосистем, таких как *DES, ГОСТ 28147-89, Blowfish*;
- стартовые значения для программ генерации ряда математических величин в асимметричных криптосистемах, например больших простых чисел в *RSA, ElGamal*;
- случайные слова, комбинируемые с парольными словами для нарушения атаки угадывания пароля криптоаналитиком;
- вектор инициализации для блочных криптосистем, работающих в режиме обратной связи;
- случайные значения параметров для многих систем электронной цифровой подписи, например *DSA*;
- случайные выборы в протоколах аутентификации, например в протоколе *Kerberos*;
- случайные параметры протоколов для обеспечения уникальности различных реализаций одного и того же протокола, например в протоколах *SET* и *SSL*.

Поскольку при реализации различных криптопреобразований используются случайные значения, стойкость криптопреобразования напрямую зависит от алгоритма формирования случайных чисел, а точнее, от их степени случайности. Современные компиляторы языков программирования обладают собственной реализацией генератора псевдослучайных последовательностей, однако с криптографической точки зрения такой генератор является непригодным.

Генератор последовательности псевдослучаен, если создаваемая им последовательность выглядит случайной, т.е. проходит все статистические тесты случайности. Последовательность называется ***криптографически безопасной псевдослучайной последовательностью***, если она непредсказуема, т.е. вычислительно неосуществимо предсказать следующий бит, имея полное знание алгоритма (или аппаратуры) и всех предшествующих бит потока.

Генератор последовательности называется ***случайным***, если он не может быть достоверно воспроизведен, т.е., если дважды запустить генератор с абсолютно одинаковыми исходными данными, получатся различные случайные последовательности.

Большинство обычных генераторов псевдослучайных чисел не подходит для использования в качестве криптографически безопасных псевдослучайных генераторов, т.к. они не обладают свойством независимости очередного генерируемого бита от предыдущего, а также ненадёжны по отношению к методам обратной разработки. ***Обратный инжиниринг*** (или ***обратная разработка***) – это исследование некоторого устройства или программы, а также документации на него с целью понять принцип его работы и, чаще всего, воспроизвести устройство, программу или иной объект с аналогичными функциями, но без копирования как такового.

Алгоритмы генерации криптографически безопасных псевдослучайных последовательностей

Алгоритм FIPS–186

Этот алгоритм является национальным стандартом США и предназначен для генерации конфиденциальных параметров и ключевой информации для национального стандартного алгоритма электронной цифровой подписи *DSA* [3]. В качестве односторонней функции G может использоваться алгоритм шифрования *DES* или алгоритм хэширования *SHA*.

Входные данные: количество генерируемых псевдослучайных чисел m и 160-битное простое число q.

Выходные данные: последовательность m псевдослучайных чисел $x_i \in \{0, 1, \dots, q-1\}$.

Шаги алгоритма:

1. Задать произвольное число b: $160 \le b \le 512$.
2. Сгенерировать некоторым образом случайное (и секретное) b-битное начальное значение s.
3. Задать вспомогательное 160-битное слово t (в шестнадцатеричной записи):

```
t = 67452301 efcdab89 98badcfe 10325476 c3d2e1f0.
```

4. Для $i = \overline{1,m}$:
   1) произвольно задать b-битное число y_i либо положить его равным нулю;
   2) вычислить $z_i = (s + y_i) \bmod 2^b$;
   3) вычислить $x_i = G(t, z_i) \bmod q$;
   4) вычислить $s = (1 + s + x_i) \bmod 2^b$.
5. Как результат выполнения предыдущего шага формируется псевдослучайная последовательность $x_1, x_2, \dots, x_m$, которую можно использовать в качестве потока псевдослучайных бит.

Алгоритм вычисления значений функции $G(t, c)$:

1. Разбить слово t на пять 32-битных слов:

$$t = H_0 \,||\, H_1 \,||\, H_2 \,||\, H_3 \,||\, H_4.$$

2. К слову c дописать справа $512 - b$ нулей так, чтобы получить 512-битное слово:

$$M = c \,||\, 0^{512-b}.$$

3. Разбить слово U на шестнадцать 32-битных слов $M = M_0 \,||\, M_1 \,||\, \dots \,||\, M_{15}$.
4. Выполнить 1 раз шаг 4 (основной цикл) алгоритма SHA–1, изменяющий H_i (т.е. для $N = 1$).
5. Выходное слово является конкатенацией:

$$G(t, c) = H_0 \,||\, H_1 \,||\, H_2 \,||\, H_3 \,||\, H_4.$$

Примечание: можно использовать стороннюю библиотеку для работы с большими числами (из 160 и более бит).

Алгоритм ANSI X9.17

Этот алгоритм является национальным стандартом США для генерации двоичной псевдослучайной последовательности [2]. Используется в приложениях, обеспечивающих безопасность финансовых платежей, и *PGP*.

В этом генераторе в качестве односторонней функции используется алгоритм шифрования *TripleDES* (*3DES*, «*тройной DES*»). Этот алгоритм использует два 64-битных ключа K_1 и K_2 следующим образом:

$$F_K(M) = E_{K_1}\left(D_{K_2}\left(E_{K_1}(M)\right)\right),$$

где $K = K_1 \,||\, K_2$ – составной 128-битный ключ, $E_{K_1}(M)$ – шифрование сообщения M алгоритмом *DES* с ключом K_1, $D_{K_2}(M)$ – дешифрование сообщения M алгоритмом *DES* с ключом $K_2 \neq K_1$.

Входные данные: некоторое случайное (и секретное) 64-битное начальное значение s_0, 128-битный составной ключ K и m – количество генерируемых 64-битных двоичных слов.

Выходные данные: последовательность m 64-битных двоичных слов $x_1, x_2, \dots, x_m$.
Схематично алгоритм представлен на рисунке 1 следующим образом:

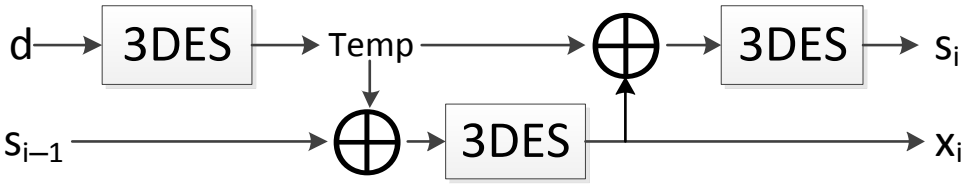

Рисунок 1 – Алгоритм ANSI X9.17

Шаги алгоритма:

1. Зафиксировать 64-битное представление d даты и времени в момент обращения к программе генерации и вычислить вспомогательное 64-битное двоичное слово $Temp = F_K(d)$.
2. Для $i = \overline{1, m}$:
   1) вычислить значение *i*-го выходного слова $x_i = F_K(Temp \oplus s_{i-1})$;
   2) вычислить новое значение параметра s_i: $s_i = F_K(x_i \oplus Temp)$.
3. В результате предыдущего шага формируется выходная псевдослучайная последовательность из m слов $x_1, x_2, \dots, x_m$ либо двоичная псевдослучайная последовательность из $64 \cdot m$ бит: $X = x_1 \,||\, x_2 \,||\, \dots \,||\, x_m$.

Примечание: алгоритм шифрования *DES* можно не реализовывать самостоятельно, а использовать готовую реализацию.

Алгоритм BBS (Blum-Blum-Shub)

Этот алгоритм основывается на проблеме факторизации больших чисел [2].
Пусть нужно сгенерировать псевдослучайную последовательность длиной m бит.

Шаги алгоритма:

1. Сгенерировать два достаточно больших секретных случайных (и различных) простых числа $p, q \equiv 3 \pmod 4$ и вычислить $N = p \cdot q$.
2. Выбрать случайное стартовое целое число s $(1 \le s \le N - 1)$ так, что $\text{НОД}(s, N) = 1$. Вычислить $u_0 = s^2 \bmod N$.
3. Для $i = \overline{1, m}$:
   1) вычислить $u_i = u_{i-1}^2 \bmod N$;
   2) вычислить x_i как самый младший бит двоичного представления числа u_i.
4. В результате предыдущего шага формируется выходная псевдослучайная последовательность $x_1, x_2, \dots, x_m$.

Примечание: можно использовать стороннюю библиотеку для работы с большими числами (из 160 и более бит).

Алгоритм RSA (Шамира)

Этот алгоритм основывается на проблеме факторизации больших чисел [2].
Пусть нужно сгенерировать псевдослучайную последовательность длиной m бит.

Шаги алгоритма:

1. Сгенерировать два различных больших простых числа p и q. Вычислить $N = p \cdot q$ и $\phi(N) = (p - 1) \cdot (q - 1)$.
2. Выбрать случайное целое число k $(1 < k < \phi(N))$, взаимно простое с $\phi(N)$, т.е. $\text{НОД}\big(k, \phi(N)\big) = 1$.
3. Выбрать случайное целое стартовое значение u_0: $1 < u_0 < N - 1$.
4. Для $i = \overline{1, m}$:
   1) вычислить $u_i = u_{i-1}^k \bmod N$;
   2) вычислить $x_i \in \{0,\ 1\}$ – самый младший бит числа u_i в двоичном представлении.

5. В результате предыдущего шага формируется выходная псевдослучайная последовательность $x_1, x_2, \dots, x_m$.

Примечание: можно использовать стороннюю библиотеку для работы с большими числами (из 160 и более бит).

Алгоритм Yarrow–160

Данный алгоритм разработан Б. Шнайером, Дж. Келси и Н. Фергюсоном в 1999 году [6]. В алгоритме для шифрования (функция $E_K()$) может использоваться алгоритм *DES* (или *3DES*), а в качестве хэш-функции $h()$ – *SHA–1*.

Шаги алгоритма:

1. Задание начальных значений:
   1) Задать размер шифруемого сообщения $n = 64$, т.к. для шифрования используется алгоритм *DES*.
   2) Задать $k = 64$ – размер ключа K, используемого при шифровании.
   3) Задать значение P_g ($0 < P_g < 2^{n/3}$, обычно $P_g = 10$), определяющее количество бит, после генерации которых нужно обновить значение ключа K.
   4) Задать значение P_t ($P_t > P_g > 0$), определяющее количество бит, после генерации которых нужно запустить механизм обновления ключа K и счётчика C_i, используя накопитель энтропии (*entropy accumulator*).
      Накопитель энтропии конкатенирует «случайные» данные (текущая дата и время, количество запущенных в системе процессов и т.п.). Пусть v – результат конкатенации накопленных данных.
   5) Задать $t = 0$, где t – количество запусков механизма обновления ключа и счётчика.
   6) Задать некоторое начальное значение n–битного счётчика C_0.
   7) Присвоить $curP_g = P_g,\ curP_t = P_t$.
2. Для $i = \overline{1, m}$ выполнить:
   1) Если $curP_g = 0$, то:
      а) с помощью функции $G(i)$ сгенерировать k бит, которые будут использоваться в качестве нового ключа K;
      б) присвоить $curP_g = P_g$.
   2) Если $curP_t = 0$, то:
      а) вычислить $v_0 = h(v \,||\, t)$;
      б) вычислить $v_i = h(v_{i-1} \,||\, v_0 \,||\, i)$ для $i = 1, \dots, t$;
      в) вычислить $K = H(h(v_t \,||\, K), k)$;
      г) вычислить $C_i = E_K(0)$;
      д) присвоить $curP_t = P_t,\ curP_g = P_g,\ t = t + 1$.
   3) Вычислить $x_i = G(i)$, которое является следующим блоком выходной последовательности.
   4) Выполнить $curP_g = curP_g - 1$ и $curP_t = curP_t - 1$.
3. В результате предыдущего шага формируется выходная псевдослучайная последовательность из m слов $x_1, x_2, \dots, x_m$ либо двоичная псевдослучайная последовательность из $n \cdot m$ бит: $X = x_1 \,||\, x_2 \,||\, \dots \,||\, x_m$.

В этом алгоритме использованы функции $G()$ и $H()$, определённые следующим образом.

Функция $G(i)$:

1. Вычислить $C_i = (C_{i-1} + 1) \bmod 2^n$.
2. Вернуть $E_K(C_i)$ как результат вычисления функции.

Функция $H(s, k)$:

1. Вычислить $s_0 = s$.
2. Вычислить $s_i = h(s_0 \,||\, \dots \,||\, s_{i-1})$ для $i = 1, 2, \dots$
3. Вернуть первые k бит от конкатенации двоичных слов $s_0 \,||\, s_1 \,||\, \dots$

Примечание: алгоритмы шифрования *DES* и вычисления значения хэш-функции можно не реализовывать самостоятельно, а использовать готовые реализации.

Статистические тесты

Существует достаточно много различных наборов тестов, оценивающих «случайность» псевдослучайной последовательности. Частично эти наборы пересекаются (одни и те же тесты встречаются в разных наборах). Наиболее известны тесты, описанные Кнутом [1], статистические тесты американского национального института стандартов и технологий *NIST* [4], батарея тестов *Diehard* [5] и др. В этой лабораторной работе будут рассматриваться некоторые тесты из набора *NIST*.

Пусть дана последовательность, состоящая из n бит. Почти все эти тесты начинаются с преобразования входной последовательности 0 и 1 (будем обозначать её ε) в последовательность -1 и 1 (будем обозначать её X) соответственно:

$$X_i = 2 \cdot \varepsilon_i - 1.$$

Частотный тест

Этот тест оценивает пропорцию нулей и единиц в проверяемой последовательности. Тест определяет, является ли количество нулей и единиц в последовательности приблизительно таким же, как должно быть в истинно случайной последовательности.

Шаги алгоритма:

1. Входная последовательность, состоящая из 0 и 1 (будем обозначать её ε), преобразовывается в последовательность -1 и 1 (будем обозначать её X) соответственно:
$$X_i = 2 \cdot \varepsilon_i - 1.$$
2. Вычисляется сумма $S_n = X_1 + X_2 + \cdots + X_n$, где n – количество элементов проверяемой последовательности.
3. Вычисляется статистика $S = \frac{|S_n|}{\sqrt{n}}$.
4. Если $S \leq 1.82138636$, то тест считается успешно пройденным, иначе делается вывод о том, что последовательность является неслучайной.

Если проверяемая последовательность не прошла данный тест, то проходить остальные тесты, проверяющие случайность, нет необходимости, т.к. уже ясно, что последовательность не является равномерно распределённой.

Тест на последовательность одинаковых бит

Этот тест анализирует количество цепочек в проверяемой последовательности, где цепочка – это непрерывная последовательность одинаковых бит. Под ***цепочкой длиной*** k понимается цепочка, состоящая из ровно k бит и ограниченная до и после битами с противоположным значением. Тест определяет, является ли количество цепочек из нулей и единиц различной длины в последовательности приблизительно таким же, как должно быть в истинно случайной последовательности.

Шаги алгоритма:

1. Вычисляется частота, с которой в проверяемой последовательности встречаются единицы:
$$\pi = \frac{1}{n} \cdot \sum_{j=1}^{n} \varepsilon_j.$$
2. Вычисляется значение $V_n = 1 + \sum_{k=1}^{n-1} r(k)$, где $r(k) = 0$, если $\varepsilon_k = \varepsilon_{k+1}$, и $r(k) = 1$ иначе.
3. Вычисляется статистика $S = \frac{|V_n - 2 \cdot n \cdot \pi \cdot (1-\pi)|}{2 \cdot \sqrt{2 \cdot n} \cdot \pi \cdot (1-\pi)}$.

4. Если $S \leq 1.82138636$, то тест считается успешно пройденным, иначе делается вывод о том, что последовательность является неслучайной.

Расширенный тест на произвольные отклонения

Этот тест оценивает общее число посещений определённого состояния при произвольном обходе кумулятивной суммы. Цель этого теста – определить отклонения от ожидаемого числа посещений различных состояний при произвольном обходе. Фактически данный тест состоит из 18 тестов, по одному для каждого состояния: $-9, -8, \dots, -1,\ 1,\ 2, \dots, 9$.

Шаги алгоритма:

1. Входная последовательность, состоящая из 0 и 1 (будем обозначать её ε), преобразовывается в последовательность -1 и 1 (будем обозначать её X) соответственно:

$$X_i = 2 \cdot \varepsilon_i - 1.$$

2. Вычисляются суммы S_i последовательно удлиняющихся подпоследовательностей, начинающихся с X_1:

$$\begin{gathered} S_1 = X_1, \\ S_2 = X_1 + X_2, \\ S_3 = X_1 + X_2 + X_3, \\ \cdot \quad \cdot \quad \cdot \\ S_n = X_1 + X_2 + X_3 + \cdots + X_n. \end{gathered}$$

3. Формируется новая последовательность $S' = 0,\ S_1,\ S_2,\ \dots,\ S_n,\ 0$.
4. Вычисляется $L = k - 1$, где k – количество нулей в полученной последовательности S'.
5. Для каждого из 18 состояний вычисляется ξ_j, которое показывает, сколько раз состояние j встречалось в последовательности S'. Здесь $j = -9, -8, \dots, -1,\ 1,\ 2, \dots, 9$.
6. Вычисляются 18 статистик $Y_j = \frac{|\xi_j - L|}{\sqrt{2 \cdot L \cdot (4 \cdot |j| - 2)}}$ для каждого состояния $j = -9, -8, \dots, -1,\ 1,\ 2, \dots, 9$.
7. Если все статистики $Y_j \leq 1.82138636$, то тест считается успешно пройденным, если же хотя бы для одной статистики Y_j это условие не выполнилось, то делается вывод о том, что последовательность является неслучайной.

Задание

I. Реализовать приложение с графическим интерфейсом, позволяющее выполнять следующие действия.
   1. Генерировать псевдослучайную последовательность с помощью заданного в варианте алгоритма:
      1) все входные параметры генератора должны задаваться из файла или вводиться в приложении;
      2) сгенерированная последовательность, состоящая из 0 и 1, должна сохраняться в файл.
   2. Проверять полученную псевдослучайную последовательность на равномерность и случайность с помощью трёх рассмотренных тестов:
      1) результат проверки каждого теста должен отображаться в приложении;
      2) все вычисляемые промежуточные значения (все шаги алгоритма теста) могут отображаться в приложении или сохраняться в файл.

II. С помощью реализованного приложения выполнить следующие задания.
   1. Протестировать правильность работы разработанного приложения.
   2. Сгенерировать последовательность из не менее 10 000 бит и исследовать её на равномерность и случайность.
   3. Сделать вывод о случайности сгенерированной последовательности и о возможности её использования в качестве криптографически безопасной псевдослучайной последовательности.

Дополнительные критерии оценивания качества работы

1. Промежуточные значения тестов:
   1 – программа отображает результаты выполнения каждого шага алгоритма каждого теста;
   0 – программа не отображает результаты выполнения хотя бы одного шага алгоритма какого-нибудь теста.

Варианты

1. Алгоритм BBS.
2. Алгоритм ANSI X9.17.
3. Алгоритм Yarrow–160.
4. Алгоритм FIPS–186.
5. Алгоритм RSA.
6. Алгоритм ANSI X9.17.
7. Алгоритм FIPS–186.
8. Алгоритм RSA.
9. Алгоритм BBS.
10. Алгоритм Yarrow–160.
11. Алгоритм ANSI X9.17.
12. Алгоритм RSA.
13. Алгоритм BBS.
14. Алгоритм Yarrow–160.
15. Алгоритм FIPS–186.

Контрольные вопросы

1. Что такое псевдослучайный генератор? Что такое криптографически безопасный генератор? Приведите примеры использования псевдослучайных чисел в криптологии.
2. Алгоритм FIPS–186.
3. Алгоритм ANSI X9.17.
4. Алгоритм BBS.
5. Алгоритм RSA.
6. Алгоритм Yarrow–160.

Список литературы

1. Кнут, Д. Получисленные алгоритмы / Д. Кнут. – М. : Мир, 1977. – 724 с. – (Искусство программирования для ЭВМ : в 3 т. ; т. 2).
2. Столлингс, В. Криптография и защита сетей: принципы и практика : Пер. с англ. / В. Столлингс. – 2-е изд. – М. : Издательский дом "Вильямс", 2001. – 672 с.
3. Харин, Ю.С. Математические и компьютерные основы криптологии : учебное пособие / Ю.С. Харин, В.И. Берник, Г.В. Матвеев, С.В. Агиевич. – Мн. : Новое знание, 2003. – 382 с.
4. A statistical test suite for random and pseudorandom number generators for cryptographic applications / NIST. – SP 800–22. Revision 1a. – Gaithersburg : NIST, 2010. – 131 p.
5. Marsaglia, G. The Diehard battery of tests of randomness [Электронный ресурс] / G. Marsaglia. – Режим доступа: http://stat.fsu.edu/pub/diehard/. – 1995.
6. Yarrow–160: Notes on the design and analysis of the Yarrow cryptographic pseudorandom number generator / J. Kelsey, B. Schneier, N. Ferguson // Yarrow 1.0. – 1999. – Vol. 1. – P. 1–14.