Федеральное агентство связи

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Сибирский государственный университет телекоммуникаций и
информатики»
(СибГУТИ)

П.А. Приставка
А.А. Ракитский

Криптографические методы защиты информации

Учебно-методическое пособие

Новосибирск
2018

УДК 002(075.8)

Утверждено редакционно-издательским советом СибГУТИ

Рецензент *канд. техн. наук, А.В. Ефимов*

Приставка П.А., Ракитский А.А. Криптографические методы защиты информации: Учебно-методическое пособие / Сибирский государственный университет телекоммуникаций и информатики; каф. Прикладной математики и кибернетики. – Новосибирск, 2018. – 53 с.

Данное учебно-методическое пособие включает в себя необходимые теоретические материалы по лабораторным работам для изучения дисциплин «Защита информации» для направления подготовки 09.03.01 «Информатика и вычислительная техника» факультета Информатики и вычислительной техники и «Методы и средства защиты информации» для направления подготовки 11.04.02 «Системы и устройства передачи данных» факультета Мультисервисных телекоммуникационных систем. В пособии рассматриваются основные элементы теории чисел, базовые алгоритмы, используемые в криптографических методах, методы шифрования данных и формирования электронной подписи. Кроме того, рассматриваются основные криптографические протоколы, используемые на практике. Также даны указания по выполнению лабораторных работ, основанных на описанных методах.

Методические указания рекомендованы для студентов технических специальностей, изучающих защиту информации на 4 курсе бакалавриата и в магистратуре.

Введение

Уровень внедрения информационных технологий в повседневную жизнь стал настолько высок, что невозможно представить хоть какую-то область деятельности человека без использования вычислительной техники. В связи с этим наиболее остро стоит вопрос защиты обрабатываемых и передаваемых данных в информационных системах. Криптографические методы защиты информации используются буквально в каждом действии при работе с различными информационными ресурсами, авторизация, организация защищённого соединения, банковские операции, передача других данных по сети и многие подобные задачи используют криптографические протоколы для обеспечения безопасности данных. В связи с этим необходимо готовить специалистов, разбирающихся в современных методах обеспечения безопасности данных и умеющих применять их на практике. Изучение дисциплин «Защита информации» и «Методы и средства защиты информации» является важной частью подготовки высококвалифицированных кадров технических специальностей. В рамках данного методического пособия рассматриваются основные элементы теории чисел и базовые алгоритмы, на которых базируется вся современная криптография. В пособии представлены методы шифрования данных, методы формирования электронной подписи и ряд важнейших протоколов, таких как «анонимное голосование», используемых на практике. Основной целью ставится изучение современных методов криптографической защиты информации и применение их на практике, так как в дальнейшем специалисты указанных направлений будут активно использовать эти методы, как в учебной, так и в профессиональной деятельности.

Алгоритм быстрого возведения числа в степень по модулю

Теоретические основы

Понятие односторонней функции

Односторонняя функция (англ. one-way function, OWF) - математическая функция, которая легко вычисляется для любого входного значения, но трудно найти аргумент по заданному значению функции. Здесь оценки «легко» и «трудно» должны пониматься с точки зрения теории сложности вычислений.

Пример:

Функция $y = ax + b$– не односторонняя функция, так как выражение $x = (b - a)/y$, которое позволяет установить аргумент по значению функции, является быстровычислимым.

В криптографических задачах в качестве односторонней функции часто используется функция $y = a^x \ mod \ p$. Обратная функция $x = \log_a x \ mod \ p$ называется дискретным логарифмом и является трудновычислимой.

Пример:

Пусть $a = 171, x = 1000000, p = 73$. Значение функции $y = 171^{1000000} mod\ 73$ равно 55 и может быть вычислено достаточно быстро (алгоритм будет рассмотрен ниже). Однако для заданных a и p установить при каком x было получено значение 55, т.е. решить уравнение

$$171^x \ mod\ 73 = 55,$$

является задачей, для которой на сегодняшний день не существует эффективных алгоритмов.

Алгоритм быстрого возведения в степень.

Одним из первых способов решения данной задачи может стать прямое умножение числа самого на себя с последующим взятием остатка от деления результата на заданное число. Программный код подобного приложения на языке C++ представлен на рис. ниже.

```
#include<stdlib.h>
#include<iostream>
#include<math.h>

using namespace std;


int main()
{
  int a,x,p,y;

  cin>>a>>x>>p;

 y=(int) pow(a,x)%p;

 cout<<y;


system("pause");
return 0;
}
```

Рис. Программный код неподходящего решения

Однако данное решение является неподходящим, как с точки зрения общей трудоемкости алгоритма (задача может быть решена за значительно меньшее количество шагов), так и с позиции корректности работы (уже для относительно небольших чисел на этапе возведения числа в степень результат операции превысит значение, которое может хранится в переменных стандартного типа языка).

Перейдем теперь к описанию алгоритма быстрого возведения числа в степень по модулю. Как хорошо известно, в информатике любое десятичное число может быть представлено в виде полинома степеней двойки.

$$x = x_0 2^0 + x_1 2^1 + \cdots + x_t 2^t,$$

где $x_t \ldots x_0$– двоичное представление числа x, а t – нижняя граница $\log_a x$.

Тогда, по свойству показательной функции

$$a^x \bmod p = \prod_{i=0}^{t} a^{x_i 2^t} \ mod\ p \qquad (1)$$

В общем виде схема вычислений может быть описана следующим образом. Сначала вычисляются значения ряда

$$a^1, a^2, a^4, \ldots, a^{2^t}\ (mod\ p), \qquad (2)$$

где каждое новое значение ряда, начиная со второго, получается путем умножения текущего члена ряда на себя. Далее, с учетом значений битов двоичного представления числа x вычисляем итоговый результат по формуле (1).

Пример.

Определить значение $5^{20}\ mod\ 7$. Сначала вычисляем значения ряда (2):

$$5^1 \quad 5^2 \quad 5^4 \quad 5^8 \quad 5^{16} \quad (mod\ 7)$$

$$5 \quad 4 \quad 2 \quad 4 \quad 2$$

Еще раз обратим внимание на способ вычисления членов ряда.

$5^2\ mod\ 7 = [(5^1\ mod\ 7) * (5^1\ mod\ 7)\ ] mod\ 7$. Так как $(5^1\ mod\ 7)$ было вычислено на предыдущем шаге, то $5^2\ mod\ 7 = 5 * 5\ mod\ 7 = 4$.

$5^4\ mod\ 7 = [(5^2\ mod\ 7) * (5^2\ mod\ 7)\ ] mod\ 7$. Так как $(5^2\ mod\ 7)$ было вычислено на предыдущем шаге, то $5^4\ mod\ 7 = 4 * 4\ mod\ 7 = 2$.

$5^8\ mod\ 7 = [(5^4\ mod\ 7) * (5^4\ mod\ 7)\ ] mod\ 7$. Так как $(5^4\ mod\ 7)$ было вычислено на предыдущем шаге, то $5^8\ mod\ 7 = 2 * 2\ mod\ 7 = 4$.

$5^{16}\ mod\ 7 = [(5^8\ mod\ 7) * (5^8\ mod\ 7)\ ] mod\ 7$. Так как $(5^8\ mod\ 7)$ было вычислено на предыдущем шаге, то $5^{16}\ mod\ 7 = 4 * 4\ mod\ 7 = 2$.

Записываем показатель степени в двоичной форме:

$$20 = 10100$$

И вычисляем итоговое значение по формуле (1):

$$y = 2 * 1 * 2 * 1 * 1\ mod\ 7 = 4$$

Рассмотрим далее трудоемкость вычислений по описанному алгоритму и покажем, что он действительно эффективнее, чем метод прямого вычисления. Для вычисления чисел ряда (2) требуется t умножений, а для вычисления результата - не более, чем t умножений. Учитывая, что $t \leq \log_2 x$, делаем вывод о том, что общая трудоемкость метода равна $2\log_2 x$ операций умножения.

В завершение темы рассмотрим псевдокод рассмотренного алгоритма:

ВОЗВЕДЕНИЕ В СТЕПЕНЬ (СПРАВА-НАЛЕВО)

ВХОД: Целые числа $a,\ x = (x_t x_{t-1} \ldots x_0)_2,\ p$.
ВЫХОД: Число $y = a^x \bmod p$.
1. $y \leftarrow 1,\ s \leftarrow a$.
2. FOR $i = 0, 1, \ldots, t$ DO
3. IF $x_i = 1$ THEN $y \leftarrow y \cdot s \bmod p$;
4. $s \leftarrow s \cdot s \bmod p$.
5. RETURN y.

На вход алгоритму поступают числа a, x и p. Требуется вычислить значение $y = a^x \ mod \ p$. Переменная y будет хранить итоговое значение произведения (1), поэтому изначально ее значение равно единице. Переменная s предназначена для хранения текущего значения ряда (2) и, следовательно, ее первым значением является a. Просматривая значения битов двоичного представления числа x от младшего к старшему, мы либо включаем в произведение множитель равный текущему члену ряда (если соответствующий бит равен 1), либо не включаем. Вне зависимости от значения бита двоичного представления на каждой итерации цикла мы вычисляем следующее значение ряда (2) путем умножения текущего ряда на самого себя. Результат вычислений будет храниться в y.

Задания для практической работы

Вычислить значения выражения

1. $y = 5^{12} \ mod \ 7$
2. $y = 3^{21} \ mod \ 11$
3. $y = 7^{31} \ mod \ 17$
4. $y = 2^{10} \ mod \ 5$
5. $y = 5^{17} \ mod \ 11$
6. $y = 3^{15} \ mod \ 10$
7. $y = 7^{12} \ mod \ 9$
8. $y = 2^{11} \ mod \ 5$

Элементы теории чисел

Теоретические основы

Простое число

Целое положительное число p называется простым, если оно имеет ровно два различных натуральных делителя – единицу и самого себя. Единица не является простым числом.

Пример.

5 и 7 – простые числа, а 4 и 9 – не простые числа.

Основная теорема арифметики

Любое целое положительное число может быть представлено в виде произведения простых чисел единственным образом.

Пример.

33=3*11, 36=3*3*4.

Взаимно простые числа

Два числа называются взаимно простыми если они не имеют ни одного общего делителя кроме единицы.

Пример.

36 и 49 – взаимно простые, 27 и 18 – не взаимно простые

Функция Эйлера

Пусть дано целое число $N \geq 1$. Значением функции Эйлера $\varphi(N)$ является количество целых чисел в ряду 1,2,3,…, N, взаимно простых с N.

Пример.

$\varphi(10) = 4$, т.к. в ряду 1, 2, 3, 4, 5, 6, 7, 8, 9 взаимно простыми с 10 будут 1, 3,7 и 9, т.е. 4 числа.

Утверждение 1

Если p - простое число, то $\varphi(p) = p - 1$

Пример.

$\varphi(11) = 10$

Утверждение 2

Пусть p и q – два различных простых числа ($p \neq q$). Тогда $\varphi(pq) = (p - 1)(q - 1)$.

Доказательство.

Рассмотрим ряд 1, 2, …, $pq-1$. В этом ряду не взаимно простыми с pq будут числа вида $p, 2p, 3p, \dots, (q-1)p$ и $q, 2q, 3q, \dots, (p-1)q$, т.е. $q-1$ чисел в первом ряду и $p-1$ чисел во втором. Вычислим теперь значение функции Эйлера:

$$\varphi(pq) = pq - 1 - q + 1 - p + 1 = (p-1)(q-1)$$

Теорема Ферма

Пусть p – простое число и $0 < a < p$. Тогда

$$a^{p-1} \bmod p = 1$$

Пример.

$2^{16} \bmod 17 = 1$, $3^{10} \bmod 11 = 1$

Теорема Эйлера

Пусть a и b - взаимно простые числа. Тогда

$$a^{\varphi(b)} \bmod b = 1$$

Пример.

$\varphi(12) = 4, 5^4 \bmod 12 = 1$

$\varphi(21) = \varphi(3*7) = 2*6 = 12, 2^{12} \bmod 21 = 1$

Утверждение 3

Если p и q – простые числа, $p \neq q$ и k – произвольное целое число, то

$a^{k\varphi(pq)+1} \bmod pq = a$

Пример.

$10^{49} \bmod 35 = 10^{2*24+1} \bmod (5*7) = 10$

Наибольший общий делитель

Пусть a и b – два целых положительных числа. Наибольшим общим делителем чисел a и b является наибольшее c, которое делит a и b. Обозначается как $\gcd(a,b) = c$.

Пример.

$\gcd(5,15) = 5, \gcd(8,28) = 4$

Алгоритм Евклида

Алгоритм Евклида служит для нахождения наибольшего общего делителя двух чисел. Псевдокод алгоритма представлен ниже.

Алгоритм АЛГОРИТМ ЕВКЛИДА

ВХОД: Положительные целые числа a, b, $a \geq b$.
ВЫХОД: Наибольший общий делитель $\gcd(a, b)$.
1. WHILE $b \neq 0$ DO
2. $r \leftarrow a \bmod b$, $a \leftarrow b$, $b \leftarrow r$.
3. RETURN a.

Пример.

Вычислить $\gcd(28,8)$.

$a = 28, b = 8$

Результаты выполнения цикла WHILE алгоритма представлены в таблице ниже.

№	r	a	b
1	28 mod 8 = 4	8	4
2	8 mod 4 = 0	4	0

Таким образом, $\gcd(28,8) = a = 4$.

Утверждение 4

Пусть a и b – два целых положительных числа. Тогда существуют целые x и y, такие что $ax + by = \gcd(a, b)$ (1)

Обобщенный Алгоритм Евклида

Обобщенный алгоритм Евклида служит для нахождения чисел x и y для заданных a и b в равенстве (1). Псевдокод алгоритма представлен ниже.

Алгоритм ОБОБЩЕННЫЙ АЛГОРИТМ ЕВКЛИДА

ВХОД: Положительные целые числа $a, b,\ a \geq b$.

ВЫХОД: $\gcd(a,b)$, x, y, удовлетворяющие (2.13).

1. $U \leftarrow (a,1,0),\ V \leftarrow (b,0,1)$.
2. WHILE $v_1 \neq 0$ DO
3. $q \leftarrow u_1 \text{ div } v_1$;
4. $T \leftarrow (u_1 \bmod v_1, u_2 - qv_2, u_3 - qv_3)$;
5. $U \leftarrow V,\ V \leftarrow T$.
6. RETURN $U = (\gcd(a,b), x, y)$.

Выходной вектор Uсодержит три элемента: на первом месте находится наибольший общий делитель чисел a и b, на втором месте - x, а на третьем – y.

Пример.

Для чисел $a = 28, b = 19$ вычислить x и y, такие что $28x + 19y = \gcd(28,19)$.

$$U = (28,1,0)$$

$$V = (19,0,1)$$

Результаты выполнения цикла WHILE алгоритма представлены в таблице ниже.

№	q	T	U	V
1	28 div 19=1	(28 mod 19, 1-1*0, 0-1*1)=(9,1,-1)	(19,0,1)	(**9**,1,-1)
2	19 div 9=2	(19 mod 9, 0-2*1, 1-2*(-1))=(1,-2,3)	(9,1,-1)	(**1**,-2,3)
3	9 div 1=9	(9 mod 1, 1-9*(-2), -1-9*3)=(0,19,-28)	(1,-2,3)	(**0**,19,-28)

Таким образом, $U = (\gcd(28,19), x, y) = (1, -2, 3)$. Проверочное равенство

$$28 * (-2) + 19 * 3 = 1$$

верно, значит задача решена.

Инверсия

Для заданных чисел c и m(c и m взаимно простые) число d ($0 < d < m$) называется инверсией числа c по модулю m, если выполняется условие

$cd \bmod m = 1$

Для обозначения числа d, которое является инверсией числа c по модулю m используется обозначение $d = \mathrm{c}^{-1} mod\ m$

Пример.

Инверсией числа $\mathrm{c} = 3$ по модулю $m = 11$ является число $d = 4$, так как выполняется условие $3 * 4\ mod\ 11 = 1$.

Для нахождения инверсии можно применять обобщенный алгоритм Евклида. Приведем далее объяснение взаимосвязи задачи нахождения инверсии и задачи, описанной в назначении обобщенного алгоритма Евклида. Из определения инверсии следует, что $cd\ mod\ m = 1$. Следовательно, $cd = km + 1$, где k - некоторое целое число. С учетом того, что c и m взаимно простые запишем:

$$(-k)m + dc = \gcd(m, c)$$

Очевидно, что для данного равенства обобщенный алгоритм Евклида позволяет вычислить $-k$ и d, если на вход будут поданы m и c. Входные вектора принимают вид

$$U = (m, 1, 0)$$

$$V = (c, 0, 1)$$

Обратим внимание, что в контексте задачи вычисления инверсии вторые элементы обоих векторов могут быть исключены, так участвуют только в определении значения $-k$. Таким образом, входными векторами будут

$$U = (m, 0)$$

$$V = (c, 1)$$

Так как инверсия числа должны быть больше нуля, то если в результате вычисления получится значение $d < 0$, необходимо прибавить к d значение m (по свойству операции mod $d\ mod\ m = d + m\ mod\ m$).

Пример.

Вычислить значение $d = 7^{-1} mod\ 11$. Определим входные вектора:

$$U = (11, 0)$$

$$V = (7, 1)$$

Результаты выполнения цикла WHILE алгоритма представлены в таблице ниже.

№	q	T	U	V
1	11 div 7=1	(11 mod 7, 0-1*1)=(4,-1)	(7,1)	(**4**,-1)
2	7 div 4=1	(7 mod 4, 1-1*(-1))=(3,2)	(4,-1)	(**3**,2)
3	4 div 3=1	(4 mod 3, -1-1*2)=(1,-3)	(3,2)	(**1**,-3)
4	3 div 1=3	(3 mod 1, 2-3*(-3))=(0,11)	(1,-3)	**0**,11)

Таким образом, $U = (\gcd(11,7), d) = (1, -3)$. Так как $d = -3$, для получения инверсии необходимо прибавить к результату m, т.е. $d = -3 + 11 = 8$. Проверяем выполнение условия из определения инверсии:

$$7 * 8 \, mod \, 11 = 1$$

Равенство верно, значит задача решена правильно.

Задания для практической работы

Вычислить значения выражения

1. $d = 3^{-1} mod \, 11$
2. $d = 5^{-1} mod \, 9$
3. $d = 7^{-1} mod \, 13$
4. $d = 6^{-1} mod \, 7$
5. $d = 3^{-1} mod \, 10$
6. $d = 5^{-1} mod \, 11$
7. $d = 4^{-1} mod \, 7$
8. $d = 2^{-1} mod \, 9$

Система Диффи-Хеллмана

Рассмотрим коммуникационную систему с большим числом пользователей. Перед нами стоит задача разработать метод, обеспечивающий защищённый канал связи между любой парой пользователей. Для решения этой задачи необходимо обеспечить обоих пользователей неким общим секретным ключом, который при этом не могут ни узнать, ни получить никакие другие пользователи. Именно на это и нацелена система Диффи-Хеллмана, она позволяет доказано-надёжным методом получить сгенерировать общий ключ, не используя для этого никаких вспомогательных средств.

В системе есть общие параметры, закрытые ключи пользователей (эта информация недоступна никому кроме самого пользователя) и открытые ключи пользователей (эта информация должна быть общедоступной). Рассмотрим все эти параметры подробнее. Для удобства будем считать, что из множества пользователей были выбраны два, Алиса и Боб, которые хотят получить общий ключ.

Общие параметры системы:

$P = 2Q + 1$, где $P, Q -$ простые числа. Число Q так же называется числом Софи Жермен, число P называют безопасным простым числом. Использование безопасных простых чисел в данной системе особенно важно, так как существуют подходы для вычисления дискретного логарифма, время работы которых зависит от величины наибольшего делителя функции Эйлера $\phi(P)$. Если P безопасное простое, мультипликативная группа чисел по модулю P имеет подгруппу высокого порядка.

$g -$ первообразный корень по модулю P. Это такое целое число, что: $g^{\phi(P)} mod\ P = 1$ и $g^l mod\ P \neq 1$, при $1 \leq l < \phi(P)$. Другими словами, первообразный корень — это образующий элемент мультипликативной группы кольца вычетов по модулю P.

Закрытые ключи: это произвольные целые числа $1 \leq X_i < P$

Открытые ключи: $Y_i = g^{X_i} mod\ P$

Алгоритм работы системы:

1. Разработчик системы генерирует общие данные P, g согласно всем описанным выше требованиям.
2. Абоненты Алиса(А) и Боб(Б) формируют свои закрытые ключи $X_A, X_Б$ и вычисляют на их основе открытые ключи $Y_A, Y_Б$.

3. Алиса вычисляет общий секретный ключ по формуле: $Z_{АБ} = Y_{Б}^{X_A} \, mod \, P$
4. Боб вычисляет общий секретный ключ по формуле: $Z_{БА} = Y_{A}^{X_Б} \, mod \, P$

Ключи $Z_{АБ}, Z_{БА}$, полученные по этому алгоритму будут равны, доказательство можно найти в [1].

При разработке такой системы необходимо в первую очередь научиться генерировать общие параметры. Обычно речь идёт о достаточно больших числах, поэтому для генерации безопасного простого числа необходимо уметь быстро проверять число на простоту. Для этого отлично подходят статистические тесты, которые позволяют утверждать, что число является простым с некоторой заданной вероятностью. Кроме того, необходимо генерировать первообразный корень, однако с учётом свойств числа P, эта задача на порядок проще.

Для генерации первообразного корня воспользуемся тем утверждением, что если $P = 2Q + 1 -$ безопасное простое число, то первообразным корнем поля по модулю P будет любое число g, для которого выполняются условия $2 \leq g < P - 1$ и $g^Q mod \, P \neq 1$. Более того, утверждается, что таких чисел достаточно много, и даже если просто перебирать все числа начиная с 2, то достаточно быстро встретится подходящее.

Методы проверки на простоту

Согласно малой теореме Ферма можно утверждать, что если для какого-то числа $a < P$ выполняется условие $a^{P-1} \, mod \, P \neq 1$, то число P является составным. Исходя из данного утверждения, можно легко разработать тест Ферма для проверки на простоту, который на языке C++ выглядит следующим образом:

```
bool testFerma(long long p, int k)
{
  if(p==2) return true;
  if(p&1) return false;
  for(int i=0; i<k; ++i)
  {
    long long a=rand()%(P-1)+1;
    if(gcd(a,p)!=1 || powMod(a,p-1,p)!=1)
      return false;
  }
  return true;
}
```

Здесь gcd(a,b) – функция нахождения наибольшего общего делителя, а powMod(a,b,p) – функция быстрого возведения в степень по модулю. Взяв случайное число a, мы проверяем утверждение: если $a^{P-1} mod\ P \neq 1$ или $НОД(P, a) > 1$, тогда P – гарантированно составное число и проверку можно прекратить. Иначе с вероятностью 50% число может быть как простым, так и составным. Таким образом, выполнив k независимых проверок, мы получим вероятность того, что число составное $\frac{1}{2^k}$. Тем не менее, важно отметить, что существуют псевдопростые числа, которые успешно проходят тест Ферма, они называются числами Кармайкла. Отчасти их можно отсечь за счёт проверки наибольшего общего делителя, но такая проверка не имеет хорошего теоретического обоснования вероятности отсечения, поэтому необходимо рассмотреть и другие тесты.

На практике чаще всего применяется тест Миллера-Рабина, он реализован во многих языках программирования и включен в стандартные библиотеки. Теоретическое описание теста можно найти, например, по ссылке: https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D1%81%D1%82_%D0%9C%D0%B8%D0%BB%D0%BB%D0%B5%D1%80%D0%B0_%E2%80%94_%D0%A0%D0%B0%D0%B1%D0%B8%D0%BD%D0%B0 . Представим здесь реализацию теста Миллера-Рабина на языке программирования C#:

```
public bool MillerRabinTest(BigInteger n, int k)
{
    if (n == 2 || n == 3)
        return true;
    if (n < 2 || n % 2 == 0)
        return false;
    // представим n-1 в виде (2^s)·t, где t нечётно, это можно
сделать последовательным делением n - 1 на 2
    BigInteger t = n - 1;
    int s = 0;
    while (t % 2 == 0)
    {
        t /= 2;
        s += 1;
    }

    for (int i = 0; i < k; i++)
    {
        // выберем случайное целое число a в отрезке [2, n-3]
        RNGCryptoServiceProvider rng = new
RNGCryptoServiceProvider();
        byte[] _a = new byte[n.ToByteArray().LongLength];
        BigInteger a;
        do
```

```
        {
            rng.GetBytes(_a);
            a = new BigInteger(_a);
        }
        while (a < 2 || a >= n - 2);

        // x ← a^t mod n, вычислим с помощью возведения в
степень по модулю
        BigInteger x = BigInteger.ModPow(a, t, n);

        // если x == 1 или x == n-1, то перейти на следующую
итерацию цикла
        if (x == 1 || x == n - 1)
            continue;

        // повторить s - 1 раз
        for (int r = 1; r < s; r++)
        {
            x = BigInteger.ModPow(x, 2, n);
            // если x == 1, то вернуть "составное"
            if (x == 1)
                return false;
            // если x == n -1, то перейти на следующую итерацию
внешнего цикла
            if (x == n - 1)
                break;
        }

        if (x != n - 1)
            return false;
    }

    // вернуть "вероятно простое"
    return true;
}
```

После выполнения k раундов теста, мы можем утверждать, что число составное с вероятностью не более $\frac{1}{4^k}$.

Шаг младенца, шаг великана

Многие шифры основываются на односторонней функции $y = a^x \bmod p$, при этом известно, что можно вычислить y, если даны a и x, затратив не более, чем $2\log_2 x$ операций. Однако отыскание x по известным a и y, т.е. вычисление дискретного логарифма — задача намного более трудная. На основании теоремы Ферма при возведении в степень по простому модулю p показатели степени приводятся по модулю $p-1$. Поэтому нам достаточно рассматривать только показатели x, удовлетворяющие неравенству $0 \le x \le p-1$. Известно, что для любых a, x, p мы можем вычислить $y = a^x \bmod p$ при помощи алгоритма быстрого возведения в степень за $O(\log_2 x)$ операций.

Теперь перейдем к задаче отыскания x по данным a, y, p. Сначала оценим сложность прямого перебора. Для этого можно было бы сначала вычислить a^1 и проверить, верно ли равенство $a^1 = y$. Если нет, то проверяем $a^2 \bmod p = y$, если нет, то $a^3 \bmod p = y$ и т.д. до $a^{p-1} \bmod p$. Количество операций линейно зависит от числа p. Таким образом, число операций, необходимое для нахождения x составляет $O(p)$. В описываемом ниже методе «шаг младенца, шаг великана» время отыскания x существенно меньше: $O(\sqrt{p}\log_2 p)$.

Данный метод впервые был описан Шенксом (Daniel Shanks) в 1973 году. Это был один из первых методов, который показал, что задача нахождения дискретного логарифма может быть решена значительно быстрее, чем методом полного перебора. Перейдём описанию метода:

1. Выбираются два целых числа m,k, такие что $mk > p$. Кроме того, сразу уточним, что для достижения минимальной трудоемкости метода нам необходимо минимизировать сумму этих чисел, что достигается при $m = k = \lceil \sqrt{p} \rceil$
2. Вычислим два ряда чисел:
$$y, ay, a^2y, \dots, a^{m-1}y \ (mod\ p)$$
$$a^m, a^{2m}, \dots, a^{km} \ (mod\ p)$$
3. Найдем такие i и j, для которых выполняется равенство
$$a^{im} = a^j y$$

Число $x = im - j$ является решением уравнения $a^x = y \ (mod\ P)$. Кроме того, такие числа i, j существуют. Доказательство можно найти в [1].

Имея оба описанных выше ряда, можно быстро найти в них равные элементы, воспользовавшись одним из описанных ниже методов.

Метод с использованием сортировки:

Пусть есть два массива $A[m], B[k]$ состоящие соответственно из m и k элементов. Ниже представлен код на языке C++, который позволяет находить в этих массивах равные элементы с трудоемкостью $O(m \log_2 m + k \log_2 k)$.

```
int A[m], B[k];
sort(A,A+m);
sort(B,B+k);
int i=0, j=0;
while(i<m && j<k)
{
  if(A[i]<B[j])
    ++i;
  else if(A[i]>B[j])
    ++j;
  else break;
}
if(i<m && j<k)
  cout << i << " " << j;
```

Необходимо учесть, что найденные индексы будут соответствовать отсортированным элементам, а для нашей задачи необходимо усовершенствовать алгоритм таким образом, чтобы при сортировке сохранялись исходные индексы, именно они будут соответствовать решению.

Существуют и другие варианты использования сортировки, можно, например, слить оба массива в один размером $m + k$, тогда после сортировки полученного массива равные элементы будут стоять по соседству. Однако, необходимо учесть сохранение исходных индексов и, кроме того, необходимо учесть, что равные элементы должны быть из разных исходных массивов (существование равных элементов внутри одного массива возможно, если a не является первообразным корнем по модулю P.

Метод с использованием структур данных:

Пусть есть два массива $A[m], B[k]$ состоящие соответственно из m и k элементов. Ниже представлен код на языке C++, который позволяет находить равные элементы с трудоемкостью $O(m \log_2 m + k \log_2 m)$. Либо, в зависимости от используемой структуры данных, с трудоемкостью $O(mC + kC)$, где C некоторая константа.

```
int A[m], B[k];
map<int,int> dict;
for(int i=0; i<m; ++i)
  dict[A[i]]=i;
```

```
for(int i=0; i<k; ++i)
  if(dict.count(B[i]))
  {
    cout << i << " " << dict[B[i]];
    break;
  }
```

Данный метод проще тем, что при его использовании не требуется учитывать исходную индексацию массивов, так как они не сортируются (хотя необходимо учесть, что второй ряд в нашей задаче индексируется от 1 до k). В зависимости от использования структуры данных на основе сбалансированных деревьев, либо на основе хэш-таблиц будет варьироваться трудоёмкость, как было указано выше. В представленном фрагменте кода используется структура map, которая базируется на красно-черных деревьях поиска.

Шифр Шамира

Теоретические основы

Шифр Шамира представляет собой средство обеспечения конфиденциального обмена сообщениями между абонентами. Рассмотрим процесс передачи секретного сообщения с помощью шифра Шамира от абонента A (Алисы) абоненту B (Бобу).

Первые действия абонентов направлены на инициализацию параметров шифра. Алиса генерирует большое просто число p и передает его Бобу по открытому каналу. Далее, Алиса формирует пару чисел C_A и D_A, так чтобы выполнялось равенство

$$C_A * D_A \; mod \; (p-1) = 1$$

Сформированная пара чисел держится Алисой в секрете. Аналогично, Боб генерирует и держет в секрете пару чисел C_B и D_B, таких, что

$$C_B * D_B \; mod \; (p-1) = 1$$

На этом формирование параметров шифра закончено и Алиса переходит к шифрованию сообщения m, представляющее собой некоторое число. Для использования данного шифра должно выполняться условие $m < p$. В случае, если сообщение не удовлетворяет этому условию, оно представляется в виде $m_0 m_1 \dots m_t$, где каждый элемент последовательности $m_i < p, 0 \le i \le t$, и каждый элемент передается отдельно. Процесс формирования криптограммы и ее последующей дешифровки включает несколько шагов.

Шаг 1. Алиса вычисляет значение

$$x_1 = m^{C_A} \; mod \; p$$

и отправляет его Бобу.

Шаг 2. Боб принимает отправленное ему число, вычисляет на его основе

$$x_2 = {x_1}^{C_B} \; mod \; p$$

и передает его Алисе.

Шаг 3. Алиса вычисляет

$$x_3 = {x_2}^{D_A} \; mod \; p$$

и отправляет его Бобу.

Шаг 4. Боб получает указанное число и вычисляет

$$x_4 = {x_3}^{D_B} \; mod \; p$$

Полученное в результате Бобом число x_4 является исходным сообщением m.

Рассмотрим доказательство. Пусть

$$e = k(p-1) + r, \text{ где } r = e \; mod \; (p-1).$$

Тогда $x^e \; mod \; p = x^{k(p-1)+r} \; mod \; p = (1^k * x^r) \; mod \; p = x^{e \; mod \; (p-1)} \; mod \; p$.

Таким образом

$$x^e \bmod p = x^{e \bmod (p-1)} \bmod p$$

Рассмотрим выражение, вычисляемое Бобом на шаге 4.

$$x_4 = {x_3}^{D_B} \bmod p = {x_2}^{D_A D_B} \bmod p = {x_1}^{D_A C_B D_B} \bmod p = m^{C_A D_A C_B D_B} \bmod p = \\ = m^{C_A D_A C_B D_B \bmod (p-1)} \bmod p = m$$

Для взлома шифра, т.е. для определения сообщения m злоумышленнику требуется знание $\mathrm{D_B}$, которое, в свою очередь, может быть получено на основе C_B. Учитывая, что x_2, x_1 и p передаются по открытому каналу, злоумышленник мог бы вычислить C_B из выражения $x_2 = {x_1}^{C_B} \bmod p$, однако для этого ему потребуется решить задачу дискретного логарифмирования, которая является очень трудоемкой.

В завершение рассмотрим пример вычислений при передаче сообщения с помощью шифра Шамира.

Пример.

Пусть Алиса хочет передать Бобу сообщение $m = 10$. Она генерирует простое число $p = 23$ и передает его Бобу. Далее, она генерирует число $\mathrm{C}_A = 7$ взаимно простое с $p - 1 = 22$ и по обобщенному алгоритму Евклида вычисляет D_A=19. Аналогично Боб вычисляет свою пару секретных чисел $\mathrm{C}_B = 5$ и $D_B = 9$. Далее происходит формирование криптограммы и ее расшифровка Бобом.

Шаг 1. $x_1 = 10^7 \bmod 23 = 14$

Шаг 2. $x_2 = 14^5 \bmod 23 = 15$

Шаг 3. $x_3 = 15^{19} \bmod 23 = 19$

Шаг 4. $x_4 = 19^9 \bmod 23 = 10$

Таким образом на четвертом шаге Бобом успешно дешифровано исходное сообщение $m = 10$.

Задания для практической работы

Для шифра Шамира с заданными параметрами C_A, C_B и p осуществить вычисление недостающих параметров и описать передачу сообщения m от Боба к Алисе.

1. $p = 23, \mathrm{C}_A = 10,\ \mathrm{C}_B = 11, m = 11$
2. $p = 31, \mathrm{C}_A = 5,\ \mathrm{C}_B = 7, m = 9$
3. $p = 17, \mathrm{C}_A = 9,\ \mathrm{C}_B = 15, m = 8$
4. $p = 29, \mathrm{C}_A = 4,\ \mathrm{C}_B = 3, m = 12$
5. $p = 19, \mathrm{C}_A = 3,\ \mathrm{C}_B = 8, m = 10$

Шифр Эль-Гамаля

Теоретические основы

Шифрование Эль-Гамаля фактически базируется на системе Диффи-Хеллмана, позволяющей сформировать общий секретный ключ у двух абонентов. Построение криптограммы представляет собой умножение исходного сообщения на указанный ключ, а дешифрование происходит путем умножения криптограммы на обратное для ключа число. Приведем далее описание процесса передачи шифрованного сообщения от абонента A (Алисы) абоненту B (Бобу).

Первоначально производится формирование общего большого простого числа p и числа $g, 1 < g < p-1$, такого, что числа $g^1 mod\, p$, $g^2 mod\, p, \ldots, g^{p-1} mod\, p$ попарно различны между собой и образуют множество $\{1,2, \ldots, p-1\}$. Более подробное описание способов выбора чисел p и g может быть найдено в разделе, посвященном системе Диффи-Хеллмана. Далее Боб генерирует целое число $x, 1 < x < p$ и производит вычисление $y = g^x\, mod\, p$. Число x хранится Бобом в секрете, а число y является открытым. Как и в шифре Шамира, преполагается, что необходимо передать сообщение $m < p$.

Шаг 1. Алиса формирует секретный сессионный ключ $k, 1 < k < p-1$

Шаг 2. Алиса вычисляет числа

$$a = g^k mod\, p$$
$$b = m * y^k mod\, p$$

и передает пару (a, b) Бобу.

Шаг 3. Боб, получает (a, b) и вычисляет

$$m' = b * a^{p-1-x}\, mod\, p$$

Значение $m' = m$, таким образом Боб получил исходное сообщение.

Для доказательства того, что $m' = m$ рассмотрим подробнее значение, вычисляемое Бобом на третьем шаге

$$m' = b * a^{p-1-1}\, mod\, p = m * y^k * g^{k(p-1-x)}\, mod\, p$$
$$= m * g^{xk} * g^{kp-k-kx} mod\, p = m * g^{k(p-1)}\, mod\, p$$
$$= m * 1^k\, mod\, p = m$$

Для того чтобы расшифровать криптограмму злоумышленнику нужно иметь секретный ключ k Алисы. Однако для вычисления k из $a = g^k mod\, p$ злоумышленник должен решить задачу дискретного логарифмирования, которая является трудновычислимой. В завершение раздела рассмотрим пример вычислений для передачи от Алисы к Бобу сообщения с помощью шифра Эль-Гамаля.

Пример.

Пусть Алиса хочет передать Бобу сообщение $m = 15$. Аналогично способу, показанному в примере вычислений параметров системы Диффи-

Хеллмана, выбирается простое число $p = 23$ и $g = 5$. Далее Боб формирует секретное число $x = 13$ и вычисляет открытое число $y = 5^{13}\ mod\ 23 = 21$.

Алиса случайным образом выбирает секретное число $k = 7$ и вычисляет пару чисел

$$a = 5^7\ mod\ 23 = 17$$

$$b = 15 * 21^7 mod\ 23 = 15 * 10\ mod\ 23 = 12.$$

Завершив вычисления, Алиса формирует криптограмму (17,12), которую передает Бобу. Боб на основе полученной пары чисел вычисляет

$$m' = 12 * 17^{23-1-13}\ mod\ 23 = 12 * 17^9\ mod\ 23 = 15$$

Таким образом, Боб получает исходное сообщение от Алисы.

Задания для практической работы

Для шифра Эль-Гамаля с заданными параметрами p, g, x и k осуществить вычисление недостающих параметров и описать передачу сообщения m от Алисы к Бобу.

1. $p = 23, g = 5,\ x = 8, m = 11$
2. $p = 23, g = 7,\ x = 10, m = 5$
3. $p = 19, g = 2,\ x = 11, m = 3$
4. $p = 17, g = 3,\ x = 5, m = 10$
5. $p = 17, g = 3,\ x = 13, m = 9$

Шифр RSA

Теоретические основы

Шифр RSA, названный в честь его разработчиков Ривеста (Rivest), Шамира (Shamir) и Адлемана (Adleman) на является востребованным и широко применимым в современной криптографии. В отличие от шифра Шамира, он лишен недостатка связанного с необходимостью многократной пересылки данных в процессе формирования криптограммы. Шифр также лишен возникающего при использовании шифра Эль-Гамаля недостатка, заключающегося в том, что размер криптограммы в два раза превышает размер исходного сообщения.

Шифр RSA базируется на так называемой односторонней функции с «лазейкой» (trapdoor function)

$$y = x^d \ mod \ p$$

Система шифрования RSA базируется на двух фактах из теории чисел:

- Низкая трудоемкость решения задачи проверки числа на простоту
- Высокая трудоемкость решения задачи разложения чисел вида $n = pq$ (p и q – большие простые) на множители (задача факторизации)

Приведем далее описание процесса передачи шифрованного сообщения от абонента A (Алисы) абоненту B (Бобу) с помощью шифра RSA.

Вначале Боб инициализирует необходимые для получения сообщений от других абонентов параметры. Он генерирует большие простые числа P и Q, и вычисляет

$$N = PQ \quad (1)$$

Далее Боб вычисляет $\phi = (P-1)(Q-1)$, выбирает $d < \phi$, такое что $\gcd(d, \phi) = 1$, и по обобщенному алгоритму Евклида вычисляет c, такое что

$$cd \ mod \ \phi = 1 \quad (2)$$

На этом этап формирования параметров шифра считается законченным. Числа N и d являются открытыми ключами Боба, а число c – его закрытым ключом. Отметим здесь, что числа P, Q и ϕ хранятся Бобом в секрете и в дальнейшем не участвуют в процессе шифрования и дешифрования сообщений. Пусть Алиса планирует передать Бобу сообщение $m < N$. Рассмотрим действия абонентов.

Шаг 1. Алиса, используя открытые ключи Боба, формирует криптограмму

$$e = m^d \ mod \ N$$

и отправляет ее Бобу.

Шаг 2. Боб принимает криптограмму и производит ее дешифровку:

$$m' = e^c \ mod \ N$$

Таким образом, вычислив m', Боб получил исходное сообщение m. Докажем это утверждение, для этого рассмотрев вычисления, проводимые Бобом.

$$m' = e^c \ mod = m^{cd} \ mod \ N$$

На основании равенства (2) имеем $cd = k\phi + 1$, где k - некоторое целое число. Согласно Утверждению 2 раздела «Элементы теории чисел»

$$\varphi(N) = \varphi(PQ) = (P-1)(Q-1) = \phi$$

Следовательно,

$$m^{cd} \ mod \ N = m^{k\phi+1} \ mod \ N = m^{k\varphi(N)+1} \ mod \ N = m$$

Учитывая, что для шифрования сообщения использовалась односторонняя функция, единственный приемлемый для злоумышленника вариант расшифровать сообщение заключается в вычислении секретного ключа Боба c. Однако, для этого злоумышленнику потребуется знание числа ϕ, т.е. ему необходимо будет разложить число N на простые множители P и Q. Как отмечалось выше, при больших P и Q задача факторизации является трундовычислимой, поэтому можно сделать вывод о том, что ширф RSA должным образом обеспечивает требование конфиденциальности.

Как отмечалось выше, функция $y = x^d \ mod \ p$ является односторонней функцией с «лазейкой». Поясним это понятие более подробно. Указанная функция является односторонней, т.е. вычисление обратной функции $x = \sqrt[d]{y} \ mod \ p$ за приемлемое время является очень трудоемкой задачей. Иными словами вычисление по известным y, d, p значения x, при котором было получено соответствующее значение функции, за приемлемое время практически невозможно. Однако, как было показано, обладание секретным ключом c, вычисленным на основе P и Q, позволяет достаточно быстро получить искомое значение аргумента. Таким образом, знание P и Q является так называемой «лазейкой» для вычисления обратной функции.

Рассмотрим теперь пример вычисления при передачи с использованием шифра RSA сообщения от Алисы к Бобу.

Пример.

Пусть Алиса хочет передать Бобу сообщение $m = 15$, который сформировал следующие параметры:

$P = 3, Q = 11$

$N = PQ = 33$

$\phi = (P-1)(Q-1) = 2 * 10 = 20$

$d = 3$, d взаимно простое с ϕ

с $= d^{-1} \ mod \ \phi = 3^{-1} \ mod \ 20 = 7$

Открытыми ключами Боба являются $d = 3, N = 33$, а закрытым ключом $c = 7$

Алиса формирует шифрованное сообщение

$$e = 15^3 \ mod \ 33 = 9$$

и передает его Бобу. Боб принимает криптограмму и дешифрует ее:

$$m' = 9^7 \ mod \ 33 = 15$$

Таким образом, Боб расшифровал исходное сообщение $m = 15$.

Задания для практической работы

Для шифра RSA с параметрами Боба P, Q и d осуществить вычисление недостающих параметров и описать передачу сообщения m от Алисы к Бобу.

1. $P = 7, Q = 11,\ d = 7, m = 11$
2. $P = 5, Q = 17,\ d = 3, m = 15$
3. $P = 3, Q = 23,\ d = 5, m = 7$
4. $P = 11, Q = 17,\ d = 3, m = 10$
5. $P = 3, Q = 11,\ d = 7, m = 12$

Шифр Вернама

Этот шифр был предложен в 1926 году американским инженером Гилбертом Вернамом и использовался на практике, но доказательство его невскрываемости было получено значительно позже Шенноном[2]. Для этого шифра также часто используется название «одноразовый блокнот». Рассмотрим этот шифр для двоичного алфавита, так как очевидно, что в случае практической реализации на компьютере, данный шифр легко может быть сведён к данному случаю. Как известно, любая информация на компьютере физически хранится и обрабатывается именно в двоичном коде.

Пусть исходное сообщение M состоит из n символов, т.е. существует всего 2^n различных сообщений M. В шифре Вернама предполагается, что ключ также состоит из n символов (количество различных ключей будет 2^n) и все ключи используются с равной вероятностью. Это означает, что вероятность появления любого ключа будет равна $P(K) = \frac{1}{2^n}$. Более того, это определение является весьма близким к определению случайной последовательности, так как все символы равновероятны и независимы.

Пусть необходимо зашифровать некоторое сообщение $m = m_1 m_2 \dots m_n$ при помощи выбранного ключа $k = k_1 k_2 \dots k_n$. Тогда зашифрованное сообщение $e = e_1 e_2 \dots e_n$ получается по формуле:

$$e_i = m_i \oplus k_i \,, \forall i = 1,2, \dots, n$$

где $\oplus$ - сложение по модулю 2. Другими словами, шифрование происходит за счет посимвольного сложения по модулю 2 сообщения и ключа. Так как сложение по модулю 2 является взаимообратной операцией, то есть $x \oplus y \oplus y = x$, то получаем, что для дешифрования сообщения необходимо применить следующую формулу:

$$m_i = e_i \oplus k_i \,, \forall i = 1,2, \dots, n$$

Докажем, что полученное в результате шифрования сообщение, так же как и ключ, будет являться случайной последовательностью. Для начала необходимо рассмотреть таблицу истинности для функции сложения по модулю 2 (побитовое исключающее или).

x	y	$x \oplus y$
0	0	0
0	1	1
1	0	1
1	1	0

Пусть вероятности сообщения известны и равны $P(m=1)=p_m, P(m=0)=1-p_m$, при этом символы ключа равновероятны, т.е. $P(k=0)=P(k=1)=\frac{1}{2}$. Тогда определим вероятности зашифрованного сообщения:

$$P(e=1)=P(k=0)P(m=1)+P(k=1)P(m=0)$$

$$P(e=0)=P(k=0)P(m=0)+P(k=1)P(m=1)$$

Подставим в формулы значения вероятностей сообщений и ключа и получим:

$$P(e=1)=\frac{p_m}{2}+\frac{1-p_m}{2}=\frac{1}{2}$$

$$P(e=0)=\frac{1-p_m}{2}+\frac{p_m}{2}=\frac{1}{2}$$

Таким образом мы доказали, что все символы зашифрованного сообщения будут равновероятны, как и символы ключа, что характеризует зашифрованную последовательность как случайную.

При выполнении работы допускается использовать любые алгоритмические генераторы псевдослучайных чисел, в том числе из стандартных библиотек, тем не менее, следует учитывать равномерность распределения символов в ключе. Т.е. при генерации случайного байта информации необходимо, чтобы его значение находилось в интервале [0,255].

Электронная подпись

Криптографическая хэш-функция

Во многих криптографических алгоритмах используются хэш-функции, обладающие определёнными свойствами, опишем подобные функции:

Криптографическая хэш-функция – это одностороннее отображение $Y: X \to H$ бесконечного множества объектов X на конечное фиксированное множество натуральных чисел H, обладающее следующими свойствами:

1) *Вычисление функции $Y = F(X)$ является относительно простой операцией и должно выполняться достаточно быстро.*
2) *Обратное вычисление $X = F^{=1}(Y)$ наоборот является крайне трудоёмкой задачей, невыполнимой для современного уровня развития вычислительной техники.*
3) *Нахождение двух разных объектов X_1, X_2, для которых значение функции будет идентичным $F(X_!) = F(X_2)$ является также крайне трудоёмкой задачей, невыполнимой при текущем уровне развития вычислительной техники.*
4) *Трудоёмкой (невыполнимой) задачей также является нахождение любого объекта $\bar{X}$, для которого выполняется равенство $F(\bar{X}) = Y$ при известном значении Y.*

Семейство международных стандартов на такие функции называется SHA (secure hash algorithm), в России существуют аналоги в виде ГОСТов. На данный момент самой надёжной функцией считается SHA3, основанная на алгоритме Keccak[3].

При выполнении лабораторных работ следует пользоваться алгоритмами с уровнем безопасности не ниже чем у алгоритма MD5. Рассмотрим подробнее именно этот алгоритм, ставший в своё время основой для первого семейства криптографических функций SHA1. Подробное описание алгоритма можно найти в [4]. Естественно при этом допускается реализовывать функцию самому, а применить любую из уже разработанных библиотек (например, openssl[5]).

Ниже приведем пример использования библиотеки openssl:

```
#include <stdio.h>
#include <string.h>
#include <openssl/sha.h>

int main() {
```

```
    unsigned char digest[SHA256_DIGEST_LENGTH];
    const char* string = "hello world";

    SHA256_CTX ctx;
    SHA256_Init(&ctx);
    SHA256_Update(&ctx, string, strlen(string));
    SHA256_Final(digest, &ctx);

    char mdString[SHA256_DIGEST_LENGTH*2+1];
    for (int i = 0; i < SHA256_DIGEST_LENGTH; i++)
        sprintf(&mdString[i*2], "%02x", (unsigned
int)digest[i]);

    printf("SHA256 digest: %s\n", mdString);


    return 0;
}
```

Здесь `SHA256_Update` используется для обновления значения хэша. Эту функцию можно использовать, например, при хэшировании файла, если выполнять чтение побайтово (или по блокам) и обновлять для каждого байта (блока) значение хэша.

Электронная подпись

Прежде чем рассматривать конкретные алгоритмы, сформулируем основные свойства, которыми должна обладать любая подпись (как электронная, так и рукописная):

1) Подписать документ от своего имени может только один человек, т.е. невозможно подделать подпись другого человека.
2) Автор подписи не может от неё отказаться, в случае споров должно быть возможно привлечение третьих сторон (например, суда) для подтверждения подлинности подписи.
3) Подпись должна защищать целостность документа и подтверждать, что подписавший его человек ознакомлен с содержимым.

На самом деле, реализовать эти свойства в электронной подписи во многом проще и удобнее, чем в физическом исполнении. Рассмотрим подробнее, как это может быть реализовано, на примере следующего алгоритма.

Уточнение по реализации лабораторных работ по теме «электронная подпись»:

В рамках лабораторной работы необходимо использовать криптографическую хэш-функцию с уровнем безопасности не ниже md5. Все

подобные функции формируют в качестве хэш-функции файла большое число размером 16/32/64 байт. Рассматриваемые ниже алгоритмы подписи предполагают работу с результатом хэш-функции как с одним числом, и это условие действительно будет выполнено, если во всех алгоритмах использовать «длинную арифметику» и работать с числами порядка 1024 бита. Тем не менее, так как требования к лабораторным работам позволяют использовать стандартные типы данных, предлагается следующий подход. Представить значение хэш-фукнции как массив байт, и каждый из этих байт подписать отдельно, применив к нему алгоритм подписи с одними и теми же изначальными параметрами (общие данные, секретный и открытый ключи).

Подпись RSA

Допустим, Алиса планирует подписать документ m. Первым делом, она должна выбрать параметры RSA, аналогично тому, как это было описано в шифре. Выбираются большие простые числа $P, Q, P \neq Q$. Вычисляются числа $N = PQ$ и $\phi(N) = (P-1)(Q-1)$. Затем Алиса выбирает число d такое, что $\text{НОД}\big(d, \phi(N)\big) = 1$ и вычисляет число c такое, что $cd\ \big(mod\ \phi(N)\big) = 1$. Наконец, Алиса публикует в открытом доступе числа N, d, и хранит в секрете остальные значения. Числа d и c будем называть соответственно открытым и закрытым ключом.

Процесс подписи документа описывается следующими шагами:

1) *Алиса вычисляет значение хэш-функции документа* $h = H(m), h < N$. *Предполагается, что алгоритм вычисления всем известен.*
2) *Алиса вычисляет число* $s = h^c (mod\ N)$.
3) *Подписанный документ* $< m, s >$ *передаётся получателю*

Каждый, кто знает открытые параметры N, d, ассоциированные с Алисой, может легко проверить подлинность подписи следующим образом:

1) *Боб получает подписанный документ <m,s> и открытые параметры Алисы N,d.*
2) *Боб вычисляет значение хэш-функции документа* $h = H(m)$.
3) *Боб вычисляет значение* $e = s^d (mod\ N)$. *В случае правильности подписи, число* $e = h$.

Указания по выполнению лабораторной работы. Согласно современным требованиям к безопасности, на используемые параметры алгоритма вводятся следующие ограничения $P, Q \approx 2^{1024}$. Но, при выполнении лабораторной работы, таких жестких ограничений нет, поэтому допускается

использовать меньшие числа, диапазон которых будет входить в стандартные типы данных. В качестве общего ограничения указано, что числа должны быть не меньше 10^9. Это означает, что в рамках данного алгоритма необходимо выбрать такие P, Q, для которых выполняется $N = PQ, N \geq 10^9$. Важно отметить, что верхней границей для числа N будет 2×10^9 (это обусловлено использованием 64-разрядного целочисленного типа данных). Таким образом, числа P, Q, чтобы их произведение попало в требуемый диапазон, будут ограничены следующим образом: $32500 \leq P, Q \leq 45000$. При использовании расширенных типов данных разрешается изменять и увеличивать границы диапазонов до описанных требованиями безопасности.

Подпись Эль-Гамаля

Пусть, как и в предыдущем случае, Алиса планирует подписывать документы. Алиса выбирает числа P, g таким образом, что $P = 2Q + 1$, при этом P, Q – простые числа, а g – первообразный корень в поле по модулю P, т.е. такое число, что $\forall\, x, y \in \{1, P-1\}, x \neq y \Rightarrow g^x \neq g^y (mod\ P)$. Алиса выбирает случайное число $1 < x < P - 1$, которое в дальнейшем будем называть секретным ключом, и вычисляет открытый ключ $y = g^x (mod\ P)$. Числа P, g, y публикуются в открытом доступе и должны быть ассоциированы с Алисой. Теперь Алиса может подписывать документы при помощи следующих шагов:

1) *Вычисляется хэш-функцию документа* $h = H(m), 1 < h < P$.
2) *Выбирается случайное число* $1 < k < P - 1$ *такое, что* НОД$(k, P - 1) = 1$, *и вычисляется число* $r = g^k (mod\ P)$.
3) *Вычисляются числа*
$$u = (h - xr)\ mod\ (P - 1),$$
$$s = k^{-1} u\ mod\ (P - 1),$$
где $k k^{-1}\ mod\ (P - 1) = 1$.
4) *Формируется подписанный документ <m,r,s>*

Для проверки корректности подписи документа выполняются следующие шаги:

1) Вычисляется хэш-функция $h = H(m)$.
2) Проверяется следующее равенство:
$$y^r r^s = g^h (mod\ P)$$

Подпись ГОСТ Р34.10-94

Так как ГОСТ является достаточно чётким указанием, помимо непосредственного описания алгоритма, необходимо учитывать, что он так же предъявляет требования к размерам чисел. Как учесть эти требования не выходя за рамки стандартных типов данных, будет предложено после описания самого стандарта.

Прежде всего, для всех пользователей системы, поддерживающей данный ГОСТ, формируются некоторые общие параметры. Выбираются два простых числа q размером 256 бит и p размером 1024 бита, которые связаны следующим соотношением:

$$p = bq + 1$$

для некоторого целого числа b. Требования к размерности чисел буквально означают, что старшие биты этих чисел должны быть равны единице. Затем выбирается число a такое, что

$$a^q \ mod \ p = 1$$

Для того, чтобы быстро получить такое число a, воспользуемся следующим приёмом. Выберём произвольное целое число $1 < g < p - 1$. Вычислим $a = g^b mod \ p$. Если $a > 1$, то мы нашли подходящий параметр, если нет, то выберем другое g. Полученное таким образом число будет гарантированно удовлетворять указанному выше требованию.

Получив три общих известных параметра p, q, a выполняется следующее. Каждый пользователь выбирает секретное число $0 < x < q$ и вычисляет:

$$y = a^x mod \ p$$

Число x – называется секретным ключом пользователя, а y – открытым ключом.

Перейдём к самому алгоритму:

1) *Абонент вычисляет хэш-функцию документа, который хочет подписать* $h = H(m)$. *Полученное значение должно лежать в пределах* $0 < h < q$.
2) *Формируется случайное число* $0 < k < q$.
3) *Вычисляем* $r = (a^k mod \ p) \ mod \ q$. *Если* $r = 0$, *то возвращаемся к шагу 2.*
4) *Вычисляем* $s = (kh + xr) mod \ q$. *Если* $s = 0$, *то возвращаемся к шагу 2.*

5) *Получаем подписанный документ* $< m, r, s >$

Для проверки подписи выполним следующие шаги:

1) *Вычисляем хэш-функцию для сообщения* $h = H(m)$.
2) *Проверяем выполнение неравенств* $0 < r, s < q$.
3) *Вычисляем* $u_1 = sh^{-1} mod\ q, u_2 = -rh^{-1} mod\ q$.
4) *Вычисляем* $v = (a^{u_1} y^{u_2}\ mod\ p) mod\ q$.
5) *Проверяем выполнение равенства* $v = r$.

Для реализации данного алгоритма в рамках лабораторной работы в его первоначальном виде, потребовалось бы использовать расширенные типы данных (так называемую «длинную арифметику»), которые поддерживаются стандартными библиотеками далеко не всех языков программирования. Поэтому предлагается следующее упрощение: пусть число p будет иметь длину 31 бит, а число q длину 16 бит. Все остальные соотношения и требования остаются прежними.

Потоковые шифры

Теоретические основы

Определение потокового шифра

Как было указано выше, совершенно секретные криптосистемы являются наилучшим способом обеспечения криптостойкости шифрограмм. Однако указанное решение обладает существенным недостатком, заключающемся необходимости передавать по закрытому каналу секретный ключ, длина которого равна длине оригинального сообщения. Для решения данной проблемы были предложены потоковые шифры, в которых в качестве ключа используется псевдослучайная последовательность – последовательность чисел, порождаемых определенным алгоритмом, однако статистически неотличимая от случайной последовательности. Алгоритмы, которые порождают псевдослучайные последовательности называются генераторами псевдослучайных чисел, а конкретный вид последовательности на выходе генератора определяется его начальным состоянием (начальными значениями переменных алгоритма). В случае использования не случайной, а псевдослучайной последовательности чисел в качестве ключа шифра можно передавать только начальное состояние генератора, так как оно позволяет независимо полностью восстановить последовательность, необходимую для дешифрования сообщения. Следует отметить, однако, что системы, использующие генераторы псевдослучайных чисел, уже не являются совершенно секретными и, в лучшем случае, требуют значительного времени для проведения процесса дешифрования (например, с помощью перебора всех возможных начальных состояний генератора).

Шифр, в котором криптограмма получается по правилу

$$y_i = x_i \oplus z_i, i = 1, \dots, n,$$

а $z_1 z_2 \dots z_n$ – псевдослучайная последовательность, называется потоковым. Дешифрование исходного сообщения осуществляется по формуле:

$$x_i = y_i \oplus z_i, i = 1, \dots, n,$$

На рис.1 ниже показано общее схема устройства потоковых шифров. Приведем необходимые пояснения.

Рис. 1. Схема устройства потоковых шифров

Пусть абонент A, Алиса, отправляет сообщение абоненту B, Бобу, с использованием потокового шифра. Каждый из абонентов обладает собственным генератором псевдослучайных чисел (ГПСЧ1 и ГПСЧ2) на своей стороне. Алиса с использованием некоторого ключа определяет начальное состояние своего генератора, который порождает ключевую последовательность $z_1 z_2 \dots z_n$, длина которой равна длине сообщения. С помощью операции сложения по модулю 2 Алиса формирует криптограмму и передает ее Бобу по открытому каналу. Для того чтобы Боб мог расшифровать полученную криптограмму ему необходимо восстановить ключевую последовательность $z_1 z_2 \dots z_n$, т.е. ему необходимо применить настройки (начальное состояние) генератора псевдослучайных чисел Алисы к своему генератору. Получить начальное состояние генератора псевдослучайных чисел от Алисы он может либо по закрытому каналу, либо по открытому каналу, с использованием, например, системы Диффи-Хеллмана. Далее, независимо сформировав псевдослучайную последовательность, Боб выполняет ее сложение по модулю 2 с криптограммой и получает исходное сообщение. Корректное дешифрование становится возможным благодаря свойству инволютивности операции сложения по модулю 2 (XOR):

$$(A \oplus B) \oplus B = \text{A}$$

Конкретный алгоритм, по которому генератор псевдослучайнх чисел формирует выходную последовательность, по сути, определяет конкретный потоковый шифр.

Для использования генератора псевдослучайных чисел в криптографических целях должно выполняться условия:

- период порождаемой последоватльности достаточно большой
- последовательность должны быть неотличима с помощью статистических тестов от случайной

В данном разделе будут подробно рассмотрено два алгоритма для генерации псевдослучайных чисел - линейный конгруэнтный генаратор и алгоритм RC4.

Линейный конгруэнтный генератор псевдослучайных чисел

Линейный конгруэнтный генератор псевдослучайных чисел является одним из наиболее простых и формирует выходную последовательность, используя правило:

$$z_{i+1} = (az_i + b) \; mod \; m,$$

где a, b, m – коэффициенты алгоритма, а z_0 - начальное значение генератор.
Пример.

Пусть коэффициенты в алгоритме линейного конгруэнтного генератора $a = 5, b = 12, c = 23$, а его начальное состояние $z_0 = 4$. Произвести вывод первых элементов порождаемой им псевдослучайной последовательности.

$$z_1 = (4 * 5 + 12)\ mod\ 23 = 9$$
$$z_2 = (9 * 5 + 12)\ mod\ 23 = 11$$
$$z_3 = (11 * 5 + 12)\ mod\ 23 = 21$$
$$z_4 = (21 * 5 + 12)\ mod\ 23 = 2$$
$$z_5 = (2 * 5 + 12)\ mod\ 23 = 22$$
$$z_6 = (22 * 5 + 12)\ mod\ 23 = 7$$
$$z_7 = (7 * 5 + 12)\ mod\ 23 = 1$$

Описанный генератор является очень слабым с точки зрения возможности использования в криптографических целях. Так, например, последовательность, порождаемая генератором может быть полностью восстановлена на основе трех первых ее членов. Указанный алгоритм лежит в основе встроенного генератора псевдослучайных чисел некоторых языков программирования (например, C++).

Опишем далее алгоритм, позволяющий восстановить параметры генератора по первым членам псевдослучайной последовательности.

Пусть $S_0, S_1, S_2, S_3, S_4, S_5, \ldots$ - первые члены порожденной линейным конгруэнтным генератором псевдослучайной последовательности. Рассмотрим разности двух ее последовательных членов:

$$t_n = S_{n+1} - S_n = (aS_n + b) mod\ m - (aS_{n-1} + b) mod\ m$$
$$= a(S_n - S_{n-1})\ mod\ m = at_{n-1}\ mod\ m$$
$$t_{n+1} = S_{n+2} - S_{n+1} = (aS_{n+1} + b) mod\ m - (aS_n + b) mod\ m$$
$$= a(S_{n+1} - S_n)\ mod\ m = a(at_{n-1}) mod\ m = a^2 t_{n-1}\ mod\ m$$

Таким образом,

$$t_{n-1} * t_{n+1} = t_n^2\ \ (mod\ m)$$

Следовательно, $t_{n-1} * t_{n+1} - t_n^2 = 0\ \ (mod\ m)$

Обозначим $u_n = |t_{n-1} * t_{n+1} - t_n^2|$. В силу того, что $u_n\ mod\ m = 0$, можно сделать вывод, что u_n кратно m, т.е.

$$u_n = k_1 m$$

Аналогичные рассуждения приводят нас к тому, что

$$u_{n+1} = k_2 m,$$

где k_1и k_2 – произвольные целые числа.

Из теории чисел известно, что вероятность того, что два числа являются взаимно простыми равна $\frac{6}{\pi^2} \approx 0.61$. Таким образом $P(\gcd(k_1, k_2) = 1) \approx 0.61$. В силу этого, $P(\gcd(k_1 m, k_2 m) = m) \approx 0.61$. Следовательно, вычисление наибольшего общего делителя от двух кратных mчисел с вероятностью большей, чем 0.5 позволит определить коэффициент m

генератора. При увеличении количества кратных m чисел, вероятность того, что их наибольший общий делить равен m тоже увеличивается, стремясь к единице.

После определения значения коэффициента m на основании первых трех элементов псевдослучайной последовательности S_0, S_1, S_2 составляется система из двух уравнений с двумя неизвестными a и b:

$$\begin{cases} S_1 = aS_0 + b \; mod \; m \\ S_2 = aS_1 + b \; mod \; m \end{cases}$$

Решение данной системы позволяет определить коэффициенты a и b генератора и восстановить полную последовательность псевдослучайных чисел.

Алгоритм RC4

Алгоритм RC4 был предложен Ривестом и предназначался специально для потокового шифрования. Алгоритм работает с -битовыми числами (словами), где n - параметр шифра. Все вычисления проводятся по модулю 2^n . Алгоритм условно можно разделить на два этапа. На первом этапе таблица упорядоченных по возрастанию чисел S $\{0, \dots, 2^n - 1\}$, перемешивается на основе состоящего из L чисел (слов) секретного ключа K. Псевдокод, соответствующий выполнению перемешивания таблицы представлен ниже:

$j \leftarrow 0, \quad S \leftarrow (0, 1, \dots, 2^n - 1);$
FOR $i = 0, 1, \dots, 2^n - 1$ DO
$\quad j \leftarrow (j + S_i + K_{i \bmod L}) \bmod 2^n,$
$\quad S_j \leftrightarrow S_i;$
$i \leftarrow 0, \quad j \leftarrow 0.$

На втором этапе выполняется генерация псевдослучайной последовательности нужной длины с попутным перемешиванием таблицы S . Каждый новый член последовательности z порождается следующим образом:

$i \leftarrow (i + 1) \bmod 2^n;$
$j \leftarrow (j + S_i) \bmod 2^n;$
$S_j \leftrightarrow S_i;$
$t \leftarrow (S_i + S_j) \bmod 2^n;$
$z \leftarrow S_t.$

Таким образом, диапазон значений порождаемых элементов равен диапазону значений чисел в таблице S и равен $[0, \dots, 2^n - 1]$.
Пример.

Пусть для алгоритма RC4 заданы следующие параметры $n = 3, K = 25$ ($L = 2$). Произвести вывод первых элементов порождаемой им псевдослучайной последовательности.

Сначала алгоритм выполняет стадию перемешивания таблицы S для определения начального состояния генератора.

$j = 0,$ начальное значение таблицы $S = \{0,1,2,3,4,5,6,7,8\}$

Результаты выполнения цикла FOR представлены в таблице ниже.

i	j	S
0	0+0+2=2	{2,1,0,3,4,5,6,7}
1	2+1+5=0	{1,2,0,3,4,5,6,7}
2	0+0+2=2	{1,2,0,3,4,5,6,7}
3	2+3+5=2	{1,2,3,0,4,5,6,7}
4	2+4+2=0	{4,2,3,0,1,5,6,7}
5	0+5+5=2	{4,2,5,0,1,3,6,7}
6	2+6+2=2	{4,2,6,0,1,3,5,7}
7	2+7+5=2	{4,2,6,0,1,3,7,5}

Вычислим несколько первых элементов элементов псевдослучайной последовательности, порождаемой генератором.

i	j	S	t	z
1	0+2=2	{4,6,2,0,1,3,7,5}	2+6=0	4
2	2+2=4	{4,6,1,0,2,3,7,5}	1+2=3	0
3	4+0=4	{4,6,1,2,0,3,7,5}	2+0=2	1
4	4+0=4	{4,6,1,2,0,3,7,5}	0+0=0	4
5	4+3=7	{4,6,1,2,0,5,7,3}	5+3=0	4
6	7+7=6	{4,6,1,2,0,5,7,3}	7+7=6	7

Задания для практической работы

Для шифра RC4 с параметрами $n = 3$ и ключом K осуществить вычисление первых 5 элементов порождаемой последовательности.

1. $K = 17$
2. $K = 11$
3. $K = 71$
4. $K = 35$
5. $K = 21$

Блочные шифры

Теоретические основы

Сеть Фейстеля (Feistel network)

Сеть Фейстеля представляет собой способ преобразования входного блока данных, который используется при построении множества современных шифров. Схемы шифрования и дешифрования входного блока данных в сети Фейстеля приводятся на рис. 1. Приведем далее словесное описание схемы.

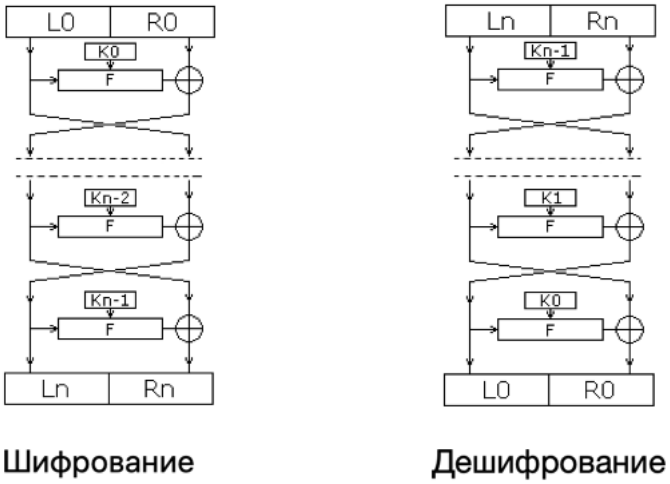

Рис. Схемы шифрования и дешифрования блока с использованием сети Фейстеля

- Входной блок разбивается на две части одинакового размера - левый (L_0) и правый (R_0).
- Левая часть L_0 изменяется некоторой функцией F с использованием раундового ключа K_0 :

$$x = F(L_0, K_0),$$

- Результат операции складывается по модулю 2 с правой частью R_0 :

$$x = x \oplus R_0$$

- Результат действия будет использован в следующем раунде в качестве левой части L_1: $L_1 = x$
- Левая часть блока текущего раунда в следующем раунде будет использована в качестве правой части R_1: $R_1 = L_0$
- Указанные операции выполняются в течение $N - 1$ раунда. На каждом раунде происходит смена раундового ключа.
- В заключительном раунде N левая часть итогового блока равна левой части блока $N - 1$ раунда. Правая часть итогового блока равна результату сложения по модулю 2 значения функции $F(L_{N-1}, K_{N-1})$ с правой частью входного блока текущего раунда.

$$L_N = L_{N-1}\ ,$$
$$R_N = R_{N-1} \oplus F(L_{N-1}, K_{N-1})$$

Конкретный вид функции F , способ формирования раундового ключ, а также количество раундов N определяются конкретным алгоритмом шифрования, который использует сеть Фейстеля для преобразования блоков. Дешифрование блока данных происходит в обратном порядке

Пример.
Дан входной блок данных, состоящий из 16-ти разрядов: 0110010111001000. Раундовая функция $F(X,k) = (X + k)\%256$, где X- левая или правая часть блока, а k – раундовый ключ. Раундовым ключом является номер раунда. Необходимо выполнить состоящее из трех раундов шифрование с помощью сети Фейстеля, а также соответствующее дешифрование.

Произведем разбиение входного блока данных на две равные части по 8 бит и перейдем для удобства в десятичную систему счисления:

$$L_0 = 01100101_2 = 100_{10},\ R_0 = \ 11001000_2 = 200_{10}$$

Приведенная ниже таблица содержит описание процесса шифрования.

№ раунда	Раундовый ключ k	Левая часть блока, L	Правая часть блока, R
1	1	$(100 + 1)\%256 \oplus 200 = 173$	100
2	2	$(173 + 2)\%256 \oplus 100 = 203$	173
3	3	203	$(203 + 3)\%256 \oplus 173 = 99$

Таким образом результатом шифрования входного блока станет последовательность $203_{10}99_{10}$ или 1100101101100011_2

Следующая таблица отображает процесс проведения дешифрования исходного блока.

№ раунда	Раундовый ключ k	Левая часть блока, L	Правая часть блока, R
3	3	$(203 + 3)\%256 \oplus 99 = 173$	203
2	2	$(173 + 2)\%256 \oplus 203 = 100$	173
1	1	100	$(100 + 1)\%256 \oplus 173 = 200$

Таким образом в результате корректного дешифрования был получен исходный блок данных 0110010111001000.

Определение блочного шифра

Блочный шифр – зависящее от ключа K обратимое преобразование блока X из n двоичных симоволов. Таким образом, если Y – это обозначение преобразованного блока, E_K – зависящая от ключа K функция

преобразования входного блока, то определение блочного шифра может быть записано как

$$Y = E_K(X)$$

Дешифрование определяется как

$$X = E_K^{-1}(Y) \text{ для всех X.}$$

Функция $\mathrm{E_K^{-1}(Y)}$ называется обратным шифром.

Функция $E_K(X)$, отвечающая за преобразование блока, может иметь различную реализацию, однако в большом количестве современных блочных шифров (например, Blowfish) указанные преобразования базируются на сетях Фейстеля, рассмотренных выше, либо на подстановочно-перестановочных сетях (например, AES). Последовательность раундовых ключей формируется на на основе ключа K с помощью определенного алгоритма.

Режимы функционирования блочных шифров

При описании преобразований на базе сети Фейстеля, лежащих в основе блочных шифров, предполагалось, что на вход поступает единственный блок. Однаков в реальных случаях сообщение разбивается на последовательность нескольких блоков, порядок обработки которых определяется режимом функционирования блочных шифров. Рассмотрим далее такие режимы функционирования блочных шифров как ECB и CBC.

- Режим ECB (Electronic CodeBook)

Исходное сообщение разбивается на последовательность блоков равной длины:

$$X = X_1, X_2, \dots, X_t$$

Далее каждый из полученных блоков независимо преобразовывается с помощью функции E_K. Преобразование осуществляется за несколько раундов с использованием в каждом раунде соответствующего раундового ключа. Т.е.

$$Y_i = E_K(X_i), 1 \le i \le t$$

Итоговая криптограмма образуется конкатенацией блоков, полученных в результате преобразований входных данных:

$$Y = Y_1, Y_2, \dots, Y_t$$

Дешифрование сообщение происходит с помощью функции обратного шифрования, т.е.:

$$X_i = E_K^{-1}(Y_i)$$

Благодаря независимой обработке блоков, для шифрования в указанном режиме можно реализовывать параллельное преобразование блоков, достигая высокой скорости формирования криптограммы. Однако

существенным недостатком режима является то, что для одинаковых блоков из X будут получены одинаковые блоки из Y, т.е. структура криптограммы будет соответствовать структуре оригинального сообщения (см. рис 2).

- Режим CBC (Cipher-Block Chaining)

Аналогично предыдущему режиму исходное сообщение разбивается на последовательность блоков равной длины:

$$X = X_1, X_2, \dots, X_t$$

Далее, на вход функции E_K последовательно подается результат сложения по модулю 2 текущего блока и результата работы функции E_K, полученного при преобразовани предыдущего блока.

$$Y_i = E_K(X_i \oplus Y_{i-1}), 1 \le i \le t$$

Если обрабатывается первый блок (т.е. результата преобразования предыдущего блока еще нет), то для получения аргумента функции E_K используется предварительно определенное значение Y_0.

Итоговая криптограмма образуется конкатенацией блоков, полученных в результате преобразований входных данных:

$$Y = Y_1, Y_2, \dots, Y_t$$

Дешифрование сообщение происходит с помощью функции обратного шифрования, т.е.:

$$X_i = Y_{i-1} \oplus E_K^{-1}(Y_i), 1 \le i \le t$$

Данный режим позволяет скрыть структуру исходного сообщения (см. рис 2), однако, в следствие последовательного преобразования блоков исходного сообщения, скорость формирования криптограммы будет ниже, чем у шифрования в режиме ECB.

Рис. 2. Результаты шифрования в режимах ECB и CBC

Задания для практической работы

Для заданных 8-битных значений L_0 и R_0, раундовой функции $F(X,k) = (X + k)\%256$, где X- левая или правая часть блока, а k – раундовый ключ, раундового ключа равного номеру раунда произвести состоящее из трех раундов шифрование входных данных и соответствующее дешифрование.

1. $L_0 = 121, R_0 = 34$
2. $L_0 = 99, R_0 = 177$
3. $L_0 = 25, R_0 = 240$
4. $L_0 = 10, R_0 = 255$
5. $L_0 = 117, R_0 = 211$

Ментальный покер

Рассмотрим задачу честной карточной игры по сети. Основной задачей в такой игре будет честное распределение карт (раздача) по игрокам без возможности мошенничества со стороны любого из них. Задача будет поставлена следующим образом.

Пусть есть колода K из $|K|$ карт (для удобства будем считать, что все карты натуральные попарно различные числа, такое сопоставление можно легко ввести для любой реальной колоды из 52 карт, если, например, сопоставить картам числа в диапазоне от 2 до 53). В игре участвуют $n \leq |K|$ игроков. Необходимо раздать каждому игроку по $m_i, 1 \leq i \leq n, \sum_{i=1}^{n} m_i \leq |K|$ карт таким образом, чтобы никто из игроков не мог знать карты другого. Более того, даже если предположить предварительный сговор между любым подмножеством игроков, они не должны иметь возможности повлиять на честность раздачи карт.

До начала игры и раздачи карт формируется большое простое число P (для большей надёжности это должно быть число Софи Жермен, то есть $P = 2Q + 1$, где P, Q - простые числа). Каждый игрок формирует два секретных ключа $1 < C_i, D_i < P - 1, 1 \leq i \leq n$, такие, что $C_i D_i \ mod\ (P - 1) = 1$. Кроме того формируется колода $K = \{k_1, k_2, \dots, k_{|K|} | k_i \in \{2, P - 1\}, k_i \neq k_j, i \neq j\}$. После получения исходных параметров можно приступить к раздаче карт, которая выполняется по следующему алгоритму:

1. Шифрование колоды:
   Каждый игрок по очереди шифрует все карты колоды при помощи своего ключа C и перемешивает их, после чего передаёт колоду следующему игроку. Опишем указанные действия для i-го игрока:
   $K_{1..i} = shuffle\left(K_{1..i-1}^{C_i}\ mod\ P\right)$
   Здесь функция $shuffle()$ случайным образом перемешивает карты в колоде (переставляет элементы массива), а $K_{1..i-1}$ – это колода, зашифрованная и перемешанная по очереди всеми игроками от 1 до $i - 1$. Так как колода K является множеством, то распишем подробнее:
   $K^C mod\ P = \{k_1^C mod\ P, \dots, k_{|K|}^C mod\ P\}$
2. После первого шага получается колода, зашифрованная и перемешанная всеми игроками по очереди. Теперь любой игрок (например, тот, который шифровал последним) может раздать всем остальным игрокам карты в любом порядке. Так как на каждом шаге осуществлялось случайное перемешивание, то карты находятся уже в

произвольном порядке и можно раздавать их подряд, в последовательном порядке, начиная с первой карты.

3. Теперь у каждого игрока на руках находятся $m_i, 1 \leq i \leq n$ карт, зашифрованных всеми игроками. Для того, чтобы узнать свои карты i-й игрок должен передать свои карты по очереди каждому игроку, чтобы те их расшифровали при помощи своих ключей D. Собственный ключ игрок использует в последнюю очередь. Опишем процесс для одной карты k i-го игрока при помощи псевдокода:

```
FOR j=1..n DO
  IF j ≠ i THEN
   k = k^Dj mod P
  FI
OD
```

$k = k^{D_i} mod\ P$

Для того, чтобы обезопасить протокол от попытки подмены карты каким-либо игроком в процессе игры, все пункты алгоритма записываются в лог, а после окончания игры каждый игрок публикует свои ключи C, D. Таким образом, любой игрок, имея записи процесса раздачи и все ключи, может проверить, что все данные корректны и ни на каком этапе не произошло подмены. Естественно, это лишь один из вариантов защиты, другие можно найти в научной литературе. Тем не менее, этот метод не нужно применять при выполнении лабораторной работы, так как он относится к протоколу лишь косвенно.

Алгоритм слепой подписи

Данный алгоритм применяется во многих криптографических протоколах, его использование для системы «электронные деньги» рассмотрено в работе [1], в данном же разделе мы рассмотрим его на примере протокола «анонимное голосование».

Опишем задачу обеспечения анонимного голосования по сети следующим образом. Пусть есть некоторый сервер, который выдаёт участникам голосования подписанные бюллетени, каждому участнику выдаётся только один бюллетень. Необходимо организовать голосование таким образом, чтобы ни сервер, ни кто-либо из участников не узнали как проголосовал конкретный участник, при этом необходимо обеспечить легитимность голосования, то есть, чтобы каждый участник мог получить только один бюллетень, который невозможно подделать и/или использовать повторно.

Опишем алгоритм голосования. Пусть есть сервер, который выдаёт бюллетени. В начале голосования сервер генерирует общие данные согласно системе RSA. Формируются два секретных больших неравных простых числа $P, Q, P \neq Q$. Вычисляется известное всем участникам число $N = PQ$. Кроме того сервер формирует секретный ключ C и открытый ключ D, связанные соотношением $CD \bmod \phi(N) = 1$, где $\phi(N) = (P-1)(Q-1)$. Очевидно, что в данном случае оба ключа должны быть числами, взаимнопростыми с $\phi(N)$. Отметим, что размеры чисел P, Q составляют 1024 бит, а числа N соответственно 2048 бит. Таким образом, всем участникам голосования известны числа D, N и только серверу известно число C. Теперь, когда участник, например, Алиса, хочет проголосовать, этот процесс будет представлять из себя следующее:

1) *Алиса формирует некоторое случайное число rnd длиной 512 бит. Далее формируется число v, которое хранит в закодированном виде результат голосования Алисы. Например, если голосование состояло только из одного вопроса с ответом Да или Нет, то закодировать результат можно как 1 или 0. Кроме того, в число добавляется некоторая служебная информация (данные о голосовании, адрес сервера и т.д.). В итоге получим некоторое число n по формуле*

$$n = rnd|v,$$

где $|$ означает конкатенацию. Необходимо учесть, что это число не должно превышать 1024 бита, таким образом под служебную информацию и результат голосования выделено 512 бит, что вполне достаточно для хранения всей информации.

2) *Алиса формирует некоторое случайное число r взаимнопростое с N. Очевидно, что вероятность случайным образом сгенерировать число, для которого не выполняется это условие крайне мала, она сопоставима с вероятностью случайным образом обнаружить делители числа N.*
3) *Алиса вычисляет криптографическую хэш-функцию от числа n. Необходимость использования этой функции мы обоснуем ниже.*
$$h = SHA3(n), h < N$$
4) *Теперь Алисе необходимо вычислить число $\bar{h} = h\, r^d (mod\ N)$. Полученное число она отправляет по защищённому верифицированному каналу связи на сервер.*
5) *Сервер помечает, что выдал бюллетень Алисе (это необходимо, чтобы Алиса не могла проголосовать дважды), и вычисляет число $\bar{s} = \bar{h}^C mod\ N$. Полученное значение отсылается обратно Алисе.*
6) *Алиса вычисляет подпись своего бюллетеня по формуле*
$$s = \bar{s}\, r^{-1} mod\ N,$$
где r^{-1} - это инверсия числа r по модулю N.
7) *Подписанный бюллетень $< n, s >$ отправляется на сервер голосования по анонимному каналу связи. Сервер проверяет корректность бюллетеня, проверив равенство $SHA3(n) = s^d mod\ N$, и, в случае корректности, учитывает голос и заносит информацию о бюллетене в открытую базу данных. Открытой она должна быть для того, чтобы любой участник мог убедиться в честности прошедшего голосования.*

Во-первых, докажем, что подпись s действительно является корректной подписью для бюллетеня n.

$$s = \bar{s}\, r^{-1} mod\ N = \bar{h}^C\, r^{-1}\ mod\ N = (h\, r^D)^C\, r^{-1}\ mod\ N$$
$$= h^C r^{DC} r^{-1}\ mod\ N$$

Здесь $r^{DC}\ mod\ N = r^1$, так как по следствию из теоремы Эйлера $r^{DC} mod\ N = r^{DC\ mod\ \phi(N)}\ mod\ N$, а $DC\ mod\ \phi(N) = 1$. Таким образом получаем

$$h^C r^{DC} r^{-1}\ mod\ N = h^C r^1 r^{-1}\ mod\ N = h^C\ mod\ N$$

Далее покажем, почему необходимо использовать криптографическую хэш-функцию, а не какую-либо произвольную $f(x) < N$. Допустим, мы использовали некоторую функцию $f(x) < N$. Предположим, что эта функция обладает мультипликативностью, т.е. $f(ab) = f(a)f(b)$. Тогда

существует возможность, имея в наличии два разных бюллетеня, получить корректный третий, используя следующие формулы:

$$n_3 = n_1 n_2$$

$$s_3 = s_1 s_2$$

Полученный таким образом бюллетень пройдет проверку на корректность, хоть и получен «нелегально». Чтобы избежать этого, необходимо выбрать такую функцию $f(x)$, которая не будет обладать ни мультипликативностью, ни аддитивностью, ни какими-либо другими свойствами, позволяющими использовать её для подделки бюллетеня, и нам известно, что криптографические хэш-функции обладают всеми необходимыми свойствами. Поэтому целесообразнее всего использовать именно их. В протоколе мы выбрали SHA3 как наиболее безопасную и защищённую от «взлома» на данный момент функцию.

Слепой подписью этот алгоритм называется потому, что сервер в момент подписывания не знает и не может узнать реального значения n и даже его значения хэш-функции, что также немаловажно, ведь криптографическая хэш-функция обладает таким свойством, что в процессе голосования крайне маловероятна ситуация, когда два разных бюллетеня будут иметь одинаковое значение хэш-функции. Это означает, что сервер мог бы, запомнив значение хэша бюллетеня конкретного человека, однозначно идентифицировать его в процессе передачи по анонимному каналу связи и связать выбранное решение с конкретным человеком, лишив тем самым процесс анонимности.

Отдельно отметим использованный в протоколе термин «анонимный канал связи», подходов для организации подобного канала существует большое число и это тема для отдельного научного исследования, поэтому, в рамках протокола, мы не будем затрагивать эту тему. Примером такого канала передачи данных является VPN, хоть он и обладает рядом недостатков с точки зрения защищённости.

В рамках выполнения задания на лабораторную работу не выдвигается требований организовывать анонимный канал связи и какую-либо работу по сети, достаточно реализовать в рамках локальной машины любой из протоколов, использующих алгоритм «слепой» подписи.

Заключение

В результате освоения данного курса обучающийся получает общие сведения об основах криптографических методах защиты информации. В ходе выполнения лабораторных работ приобретаются навыки практического использования наиболее используемых криптографических протоколов и формируются навыки построения аналогичных протоколов для более конкретных задач. В дальнейшем полученные навыки позволяют понимать более сложные методы защиты информации, сформированные на основе эллиптических кривых и применяющиеся в современных информационных система. Знание устройства подобных методов помогает, в том числе, лучше встраивать их в существующие информационные системы и обнаруживать уязвимости в уже реализованных решениях.

Список литературы

1. *Б.Я. Рябко, А.Н. Фионов. Криптографические методы защиты информации. 2-е издание. Москва. Горячая линия – Телеком. 2012*
2. *Шеннон К. Работы по теории информации и кибернетике. – М.: ИЛ, 1963. – С. 333–369 (Теория связи в секретных системах).*
3. *Bertoni, G., Daemen, J., Peeters, M., & Van Assche, G. (2013, May). Keccak. In Annual international conference on the theory and applications of cryptographic techniques (pp. 313-314). Springer, Berlin, Heidelberg.*
4. *Rivest, Ronald. The MD5 message-digest algorithm. No. RFC 1321. 1992.*
5. *Open library for secure protocols. https://www.openssl.org/*

Содержание

Учебное издание

Приставка Павел Анатольевич
Ракитский Антон Андреевич

Криптографические методы защиты информации

Редактор *А.В. Ефимов*
Корректор

Подписано в печать 21.11.2018.
Формат бумаги 62 × 84/16, отпечатано на ризографе, шрифт № 10,
п. л. – , заказ № , тираж – 500.

Редакционно-издательский отдел СибГУТИ
630102, г. Новосибирск, ул. Кирова, 86, офис 107, тел. (383) 269-82-36