

Лабораторная работа № 2. Изучение алгоритма RSA

Цель работы:

Освоить механизм шифрования и дешифрования данных в криптографической системе с открытыми ключами RSA.

Постановка задачи:

1. Разработать программу, осуществляющую шифрование и дешифрование сообщения алгоритмом RSA. Ключи генерируются на основе чисел p и q , значения которых выбирается в соответствии с вариантом. При выборе числа e использовать минимально возможное.
2. Исходное сообщение M может состоять из символов как русского, так и любого другого алфавита.
3. Обеспечить вывод ключей и зашифрованного текста.
4. В программе предусмотреть проверку, являются ли два числа взаимно простыми.
5. Результаты работы оформить в виде отчета.

Описание используемого метода

1. Выбирается два больших простых числа p и q .
2. Определяется $n = p * q$.
3. Выбирается большое случайное число e . Это число должно быть взаимно простым с результатом $(p - 1) * (q - 1)$.
4. Определяется такое число d , для которого является истинным соотношение $(e * d) \bmod ((p - 1) * (q - 1)) = 1$.
5. Открытым ключом являются числа e и n , а секретным ключом - числа d , p и q .
6. После произведенного выбора открытого и секретного ключей для шифрования данных необходимо выполнить следующие действия:
 - разбить шифруемый текст на блоки, каждый из которых может быть представлен в виде числа $M_i = 0, 1, \dots, n - 1$;
 - зашифровать текст, рассматриваемый как последовательность чисел $M(i)$ по формуле: $C_i = M_i^e \bmod n$.
7. Чтобы расшифровать эти данные, используется секретный ключ $\{d, n\}$ и выполняются следующие вычисления: $M_i = C_i^d \bmod n$.
8. В результате получают исходный текст M_i .

Описание исходных данных:

В данной работе я выбираю 1 вариант, поэтому у меня числа p и q будут иметь следующие значения: $p = 193$, $q = 353$.

Исходный текст:

"Every complex problem has a solution that is clear, simple, and wrong. - H. L. Mencken"

Алгоритм работы программы:

Следует заметить, что в качестве вспомогательных алгоритмов, я использую расширенный алгоритм Евклида (см., например, здесь

https://en.wikipedia.org/wiki/Extended_Euclidean_algorithm) и алгоритм быстрого возведения в степень по модулю.

Расширенный алгоритм Евклида - это модификация алгоритма Евклида, вычисляющая, кроме наибольшего общего делителя (НОД) целых чисел a и b , ещё и коэффициенты соотношения Безу, то есть такие целые x и y , что

$$a * x + b * y = \text{НОД}(a,b).$$

Алгоритм быстрого возведения в степень по модулю позволяет возводить число в некоторую степень (по некоторому модулю) со сложностью $O(\log N)$, используя двоичное представление степени.

1. Проверяем, что числа p и q взаимно простые.
2. Вычисляем значение n ($n = p * q$)
3. Вычисляем минимально возможное значение числа e , взаимно простое со значением $(p-1)*(q-1)$. Т.к. числа p и q простые, то значение $(p-1)*(q-1)$ четное (это очевидно); значит значение числа e должно быть нечетным. Мы в цикле проверяем значение e на взаимную простоту со значением $(p-1)*(q-1)$, начиная с числа 3 и увеличивая каждый раз его на 2; как только значение e будет найдено - это будет минимально возможное значение для e .
4. Вычисляем значение числа d с помощью соотношения $(e*d) \bmod ((p-1)*(q-1)) = 1$. Для этого мы используем расширенный алгоритм Евклида.
5. Мы получили ключи для нашей реализации алгоритма RSA: открытым ключом являются числа e и n , закрытым ключом - числа d , p и q .
6. Разбиваем шифруемый текст на блоки, каждый из которых может быть представлен в виде числа $M_i = 0, 1, \dots, n-1$. Для этого мы преобразуем строку в массив байт в кодировке UTF-8; этот массив байт преобразуем в одно большое число (так, например, если у нас массив байт состоит из 2 байт - 0x66 и 0x55, то результирующее число будет иметь следующее значение - 0x5566); после чего это число из системы счисления по основанию 10 преобразуем в число в системе счисления по основанию n (стандартным для этого подходом).
7. Кодлируем полученный массив блоков - для этого каждый блок преобразуем по следующей формуле: $C_i = M_i^e \bmod n$ (т.е. с помощью открытого ключа).
8. Получаем закодированный набор данных. Выводим его на экран.
9. Декодируем закодированный массив блоков - для этого каждый блок преобразуем по следующей формуле: $M_i = C_i^d \bmod n$ (т.е. с помощью закрытого ключа).
10. Получаем декодированный набор данных.
11. Преобразуем блоки, каждый из которых может быть представлен в виде числа $M_i = 0, 1, \dots, n-1$, в массив байт. Блоки являются цифрами некоторого большого числа в системе счисления по основанию n ; вычисляем значение этого числа в системе счисления по основанию 10; после чего преобразуем это число в массив байт (если число будет иметь следующее значение - 0x5566, то мы получим массив из 2 байт - 0x66 и 0x55). Преобразуем массив байт в строку с помощью кодировки UTF-8.
12. Выводим полученную строку на экран
13. Сравниваем исходный массив блоков с зашифрованным массивом блоков - они должны не совпадать.
13. Сравниваем исходный массив блоков с декодированным массивом блоков - они должны совпадать.

Тексты программы:

В качестве языка программирования я использую C#. Исходный текст программы приведен в отдельном файле Program.cs (без дополнительных файлов проекта - *.csproj и решения - *.sln).

Результаты работы программы:

Приватный ключ:

$p = 193$, $q = 353$, $d = 13517$

Публичный ключ:

$n = 68129$, $e = 5$

Исходный текст:

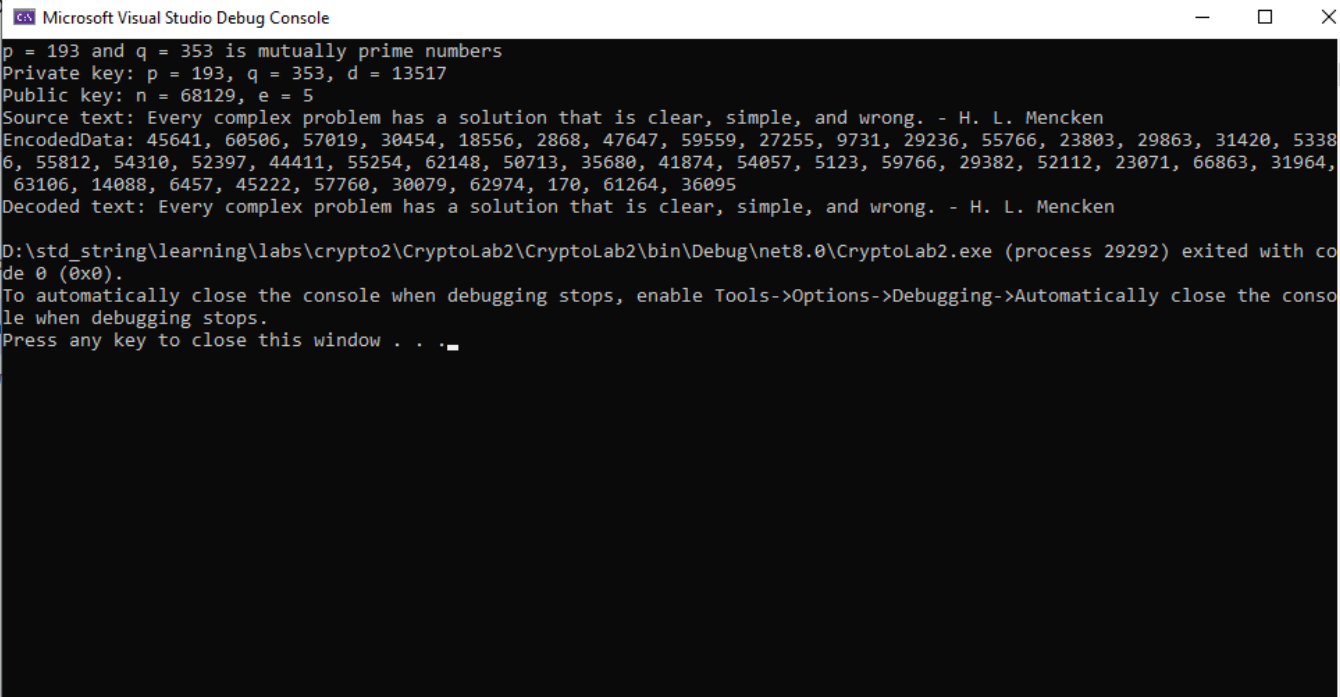
Every complex problem has a solution that is clear, simple, and wrong. - H. L. Mencken

Закодированные данные:

45641, 60506, 57019, 30454, 18556, 2868, 47647, 59559, 27255, 9731, 29236, 55766, 23803, 29863, 31420, 53386, 55812, 54310, 52397, 44411, 55254, 62148, 50713, 35680, 41874, 54057, 5123, 59766, 29382, 52112, 23071, 66863, 31964, 63106, 14088, 6457, 45222, 57760, 30079, 62974, 170, 61264, 36095

Декодированный текст:

Every complex problem has a solution that is clear, simple, and wrong. - H. L. Mencken



```
Microsoft Visual Studio Debug Console
p = 193 and q = 353 is mutually prime numbers
Private key: p = 193, q = 353, d = 13517
Public key: n = 68129, e = 5
Source text: Every complex problem has a solution that is clear, simple, and wrong. - H. L. Mencken
EncodedData: 45641, 60506, 57019, 30454, 18556, 2868, 47647, 59559, 27255, 9731, 29236, 55766, 23803, 29863, 31420, 53386, 55812, 54310, 52397, 44411, 55254, 62148, 50713, 35680, 41874, 54057, 5123, 59766, 29382, 52112, 23071, 66863, 31964, 63106, 14088, 6457, 45222, 57760, 30079, 62974, 170, 61264, 36095
Decoded text: Every complex problem has a solution that is clear, simple, and wrong. - H. L. Mencken

D:\std_string\learning\labs\crypto2\CryptoLab2\CryptoLab2\bin\Debug\net8.0\CryptoLab2.exe (process 29292) exited with code 0 (0x0).
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```

Анализ результатов:

1. С помощью алгоритма RSA мы можем как закодировать, так, потом и декодировать произвольные данные.
2. Легко видно, что алгоритм RSA является асимметричным алгоритмом: публичный ключ, который используется для шифрования данных (в нашем примере - это n =

68129, $e = 5$) не совпадает с приватным ключом, который используется для декодирования (в нашем примере - это $p = 193$, $q = 353$, $d = 13517$)

3) Из реализации видно, что производительность (скорость работы) алгоритма RSA достаточно не высока: он содержит такие ресурсоемкие алгоритмы, как возведение большого числа в большую степень, разбиение исходных данных на массив блоков (для шифрования), преобразование массива блоков в исходные данные (для декодирования). К тому же, чем больше обрабатываемые данные, тем сложнее задача по шифрованию и декодированию; именно поэтому обычно асимметричные алгоритмы используют для передачи ключа симметричных алгоритмов шифрования (которые гораздо более производительны).

Выводы:

Мы изучили и реализовали алгоритм RSA. Мы использовали его для шифрования и декодирования данных; видно, что мы можем зашифровать некоторые данные с помощью алгоритма RSA, а потом декодировать и мы получим те же самые данные. Также мы увидели, что алгоритм RSA очень чувствителен к качеству реализации некоторых алгоритмов; так например, если быстрое возведение в степень заменить на обычное, то скорость работы алгоритма RSA упадет в разы.

Контрольные вопросы:

1. Что такое однонаправленные функции?

Односторонняя (однонаправленная) функция — математическая функция, которая легко вычисляется для любого входного значения, но трудно найти аргумент по заданному значению функции. Здесь «легко» и «трудно» должны пониматься с точки зрения теории сложности вычислений. Разрыв между сложностью прямого и обратного преобразований определяет криптографическую эффективность односторонней функции. Неинъективность функции не является достаточным условием для того, чтобы называть её односторонней. Односторонние функции могут называться также трудно обратимыми или необратимыми. Ни для одной функции, используемой как односторонняя на практике, не доказано, что она истинно односторонняя (то есть, что не существует быстрого алгоритма решения обратной задачи). Такое доказательство докажет, что классы сложности P и NP не равны, попутно разрешив ряд вопросов теоретической информатики. Современная асимметричная криптография основывается на предположении, что односторонние функции всё-таки существуют.

2. Основные свойства однонаправленных функций с потайным ходом.

Односторонняя функция с потайным входом (англ. trapdoor function, TDF) — это односторонняя функция f из множества X в множество Y , обладающая свойством (потайным входом, лазейкой), благодаря которому становится возможным найти для любого $y \in Imf$, $x \in X$ такое, что $f(x) = y$, то есть обратить функцию. Односторонние функции с потайным входом широко используются в асимметричных методах шифрования (шифрование с открытым ключом) таких как: RSA. В настоящее время доподлинно не установлено, что односторонние функции с потайным входом

действительно являются односторонними, то есть нет доказательства того, что, не зная потайной вход, криптоаналитик не сможет обратить функцию.

Пример односторонних функций с потайным входом; RSA:

Возьмём число $n = p * q$, где p и q принадлежат множеству простых чисел. Считается, что для данного числа n вычисление p и q является математически трудной задачей.

Функция шифрования RSA — $E(m) = m^e \bmod n$, где e — взаимно простое число с $(p - 1) * (q - 1)$. Числа p и q являются потайным входом, зная которые вычислить обратную функцию E^{-1} легко. На сегодняшний день лучшей атакой на RSA является факторизация числа n .

3. Какие числа называются взаимно простыми?

Взаимно простыми числами называются такие числа, которые не имеют общих делителей, кроме единицы. Это значит, что наибольший общий делитель (НОД) таких чисел равен 1.

4. Как реализуется программное возведение в степень для больших чисел?

Стандартный подход возведения числа a в степень n - это $(n - 1)$ умножение числа a на результат. Он имеет линейную сложность $O(N)$. Существуют несколько вариантов быстрого возведения числа в степень с логарифмической сложностью $O(\log N)$ - см., например, здесь https://en.wikipedia.org/wiki/Exponentiation_by_squaring. Например, алгоритм по схеме "справа налево" имеет следующий вид:

Пусть $n = (m_k m_{k-1} \dots m_0)_2$ - двоичное представление степени n .

Последовательность действий при реализации данного алгоритма следующая:

1. Представить показатель степени n в двоичном виде.
2. Положить вспомогательную переменную z равной числу x .
3. Если $m_i = 1$, то текущий результат умножается на z , а само число z возводится в квадрат. Если $m_i = 0$, то требуется только возвести z в квадрат. При этом индекс i изменяется от 0 до $k-1$ включительно.

5. На чем основана криптостойкость алгоритма RSA?

Надежность (криптостойкость) алгоритма RSA основана на трудноразрешимой задаче разложения n на сомножители (то есть на невозможности факторинга n , который относится к классу NP задач) так как в настоящее время эффективного способа поиска сомножителей не существует.

6. Каковы достоинства и недостатки асимметричных алгоритмов?

К преимуществам асимметричных алгоритмов можно отнести удобное распределение открытых ключей, которое не требует никакой секретности. К недостаткам - невысокую скорость работы с открытым ключом (из-за этого асимметричные концепции применяются относительно небольших по размеру данных); длину ключей - она оказывается на порядок больше, чем в симметричных криптосистемах.