**Лабораторная работа №4**

*Тема: Криптосистема* [*RSA*](https://ru.wikipedia.org/wiki/RSA)*.*

*Введение.* Формально, как и для любой [ассиметричной криптосистемы](https://ru.wikipedia.org/wiki/%D0%9A%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D1%81_%D0%BE%D1%82%D0%BA%D1%80%D1%8B%D1%82%D1%8B%D0%BC_%D0%BA%D0%BB%D1%8E%D1%87%D0%BE%D0%BC), для работы RSA необходимо реализовать 3 основные процедуры: генерацию ключей, функцию зашифрования и функцию расшифрования. Подробности можно посмотреть в [лекции](https://drive.google.com/open?id=1UZJH3U1W5FGzK98hmC9JzcybXxsdBQgd).

*Генерация ключей (Gen).*

1. Выбираются два больших простых числа *p* и *q*.

2. Вычисляется число *n* = *pq*; *n* –модуль.

3. Открытый ключ *e* выбирается из *Zϕ(n) = {0, ..., ϕ(n)-1}* таким образом, чтобы *НОД (e, ϕ(n)) = 1* (очевидно, что зная *p* и *q,* можно легко вычислить *ϕ(n) = (p-1)(q-1)*).

4. Личный ключ *d* из *Zϕ(n)* находится по формуле *d = e-1 (mod ϕ(n)).*

*Функция зашифрования (Encr).*

*Ee(X) = Xe (mod n)*, где *n* –модуль, *e* – открытый ключ.

*Функция расшифрования (Decr).*

*Dd(Y) = Yd (mod n)*, где *n* –модуль, *d* –личный ключ.

*Замечания:*

* Для возведения в степень рекомендуется использовать [алгоритм быстрого возведения в степень](http://e-maxx.ru/algo/binary_pow). Нельзя использовать готовую реализацию этого алгоритма, если она есть в языке программирования, на котором вы выполняете лабораторную работу.
* Проверить, что случайно выбранное число *e* удовлетворяет требуемому условию *НОД(e, ϕ(n)) = 1*, и, если условие выполнено, найти личный ключ *d* = *e*-1 (mod ϕ(*n*)) можно с помощью [расширенного алгоритма Евклида](http://e-maxx.ru/algo/export_extended_euclid_algorithm). Нельзя использовать готовую реализацию этого алгоритма, если она есть в языке программирования, на котором вы выполняете лабораторную работу.
* Сообщение *X*, которое необходимо зашифровать, и сообщение *Y,* которое необходимо расшифровать, представляются в виде некоторого числа из *Zn*.

*Условие лабораторной работы:*

* Для выполнения всех заданий необходимо использовать "длинную" арифметику. Разрешается использовать любую готовую библиотеку или написать свою. При этом учитывайте, что допустимые языки программирования {С/С++, C#, Java, Python, R}.
* Ваша программа должна принимать на вход указание, какую операцию необходимо выполнить: *Gen*, *Encr*, *Decr или Exit*. В зависимости от операции программа должна считывать (из консоли, файла и т.п.) необходимые данные:
* *Gen*: *p*, *q* и *e;*
* *Encr*: *X*, *e* и *n*;
* *Decr: Y*, *d* и *n:*
* *Exit*: нет параметров.
* На выход (в консоль, файл и т.п.) программа возвращать результат выполненной операции: *d* для *Gen*, *Y* для *Encr*, *X* для *Decr.* При вызове операции *Exit* программа должна завершать свою работу.
* Программа должна иметь проверку корректности входных данных (для *Gen,* чтоНОД(*e*, *ϕ(n)*) = 1; для *Encr* и *Decr,* что *X* и *Y* соответственно лежат в Z*n*).
* Программа должна предлагать пользователю выполнить очередную операцию до тех пор, пока пользователь не выберет вариант выхода из программы.

*Отчет (*условия по вариантам смотрите на последней странице)*:*

1) В условиях своего варианта для заданных чисел *p*, *q* и *e*, необходимо вычислить личный ключ *d*.

2) Для заданного сообщения *X*1, вычислить зашифрованное сообщение *Y*1, используя открытый ключ *e*.

3) Расшифровать сообщение *Y*1, используя личный ключ *d*, сравнить результат с исходным сообщением *X*1.

4) Для заданного шифртекста *Y*2, вычислить исходный открытый текст *X*2, используя личный ключ *d*.

5) Результаты шагов 1 – 4 необходимо записать в текстовый файл, назвав его «Report.txt».

*Бонусные задания.*

**1)** Реализовать один из вероятностных методов проверки числа на простоту:

* [Тест Соловея-Штрассена](https://ru.wikipedia.org/wiki/Тест_Соловея_—_Штрассена);
* [Тест Миллера- Рабина](https://ru.wikipedia.org/wiki/Тест_Миллера_—_Рабина).

Данные тесты, проверяют, является ли число составным. Если хотя бы на одной итерации тест скажет, что число составное, то оно действительно составное, если же все итерации пройдут, то с вероятностью близкой к 1 (зависит от теста и числа итераций) можно утверждать, что число простое. Умея проверять число на простоту легко генерировать относительно большие простые числа: выбираем случайное нечетное число *n* требуемого нам размера (в битах), проверяем его на простоту, если оно простое, то все хорошо, если же нет, то берем число *n*+2 и т.д.

Реализация должна принимать на вход желаемый размер простого числа (в битах), а на выходе возвращать простое (с вероятностью не ниже 0.999999) число заданного размера. Алгоритм должен использовать "длинную" арифметику.

**2)** Реализовать один из методов факторизации числа на делители, отличный от полного перебора делителей (простых делителей) от 2 до корня из числа. Возможные алгоритмы факторизации можно найти [здесь](https://ru.wikipedia.org/wiki/Факторизация_целых_чисел).

Реализация должна принимать на вход простое число, а на выходе возвращать его разложение на простые множители. Гарантируется, что разложение будет состоять ровно из двух простых множителей, которые будет иметь размер, позволяющий осуществить факторизацию за разумное время. Алгоритм должен использовать "длинную" арифметику.

**3)** Реализовать атаку повторного шифрования на алгоритм RSA (см. материалы [лекции](https://drive.google.com/open?id=1UZJH3U1W5FGzK98hmC9JzcybXxsdBQgd) или лекцию на [Интуит](https://www.intuit.ru/studies/courses/13837/1234/lecture/31200?page=3)).

Реализация должна принимать на вход *X*, *e* и *n*, а на выходе возвращать *Y*. Гарантируется, что *n* будет иметь размер, позволяющий осуществить атаку за разумное время. Алгоритм должен использовать "длинную" арифметику.

**Замечание**: Успешная реализация любого из алгоритмов проверки на простоту или факторизации, а также атаки повторного шифрования дает возможность НЕ выполнять 0.5 (половину) любой из 5 (пяти) основных лабораторных работ. Можно реализовать несколько алгоритмов: все бонусы суммируются (т.е., если вы реализуете 2 (два) алгоритма проверки числа на простоту, 1 (один) алгоритм факторизации и атаку повторного шифрования, то сможете НЕ выполнять 2 (две) из 5 (пяти) основных лабораторных работ).

*Порядок сдачи лабораторной работы:*

* Вам необходимо создать архив формата «.zip», название которого должно иметь вид «4\_5\_Ivanov.zip», где «5» – № вашей группы, а «Ivanov» – ваша фамилия латинскими буквами. В архиве должен быть исходный код вашей реализации двух классических шифров (НЕ весь проект целиком, а только файлы с исходным кодом (.c, .cpp, .java и т.п.)) и файл «Report.txt»;

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **№ вар.** | ***p*** | ***q*** | ***e*** | ***X*1** | ***Y*2** |
| 1 | 563036103490583 | 1063300642915937 | 372585779765210097553647509959 | 399754188907643924420059310699 | 293314580135454643114146935352 |
| 2 | 684391453787369 | 938396705691661 | 245372344253915653531369256899 | 184712154522842417799563173273 | 447204864183801463638208868116 |
| 3 | 950226133300007 | 973747816218557 | 272205786540380931859823391349 | 487590396324873679144487947752 | 371209390170967767404608751313 |
| 4 | 882493304303057 | 565640080106113 | 435510454193522616856570224823 | 33938304564942541056706890572 | 167669363821217143128176537107 |
| 5 | 801410357975153 | 950867021741191 | 110066171603901969362593059313 | 651256495894733822754552878879 | 253970845268857814403399216528 |
| 6 | 877624729981871 | 977289103272413 | 393858705445953556259457155603 | 99213684276131904144063593842 | 3323645942935447942371627841 |
| 7 | 578569278720973 | 976534805568533 | 235108486320061234453015373083 | 402700874043636335474593885222 | 416593343738152120120255791792 |
| 8 | 1102914252601991 | 571301412050021 | 624840313709071966800768010501 | 267222621555915275276288463243 | 291064433434228628162063527294 |
| 9 | 1044779951622553 | 810317495045789 | 657311140049004998195550456065 | 543066057187844905915287549311 | 482438945755863228767362999535 |
| 10 | 749491517671339 | 1074991257344881 | 224872067131777233484376270407 | 469827559223869083025116205997 | 135039041969537192266152584876 |
| 11 | 1003636526287921 | 630114564355681 | 227175758417350578804323023499 | 94380455332205669620317800575 | 493089842126272405947077785493 |
| 12 | 630771436373581 | 786871162464589 | 145903195239272744119599032753 | 153836541207562724815850112591 | 154774251062114172643840396063 |
| 13 | 1101233370547069 | 913071788602213 | 441294907009469893617176298995 | 832192044845038413443817859011 | 381868705201087633178499417569 |
| 14 | 804288300171659 | 819139104388459 | 587862009679843002844824189377 | 201229993267158788910642144722 | 474981332560572636448191786787 |
| 15 | 722227767914309 | 645306156219341 | 188343213856435087990117867713 | 340693830559923670446313718411 | 401641252598150824512742688568 |

Таблица №1: Тестовые варианты параметров алгоритма RSA