Лабораторная работа № 7

Реализация алгоритма ассиметричного шифрования RSA в соответствии с семейством стандартов PKCS

Введение

Шифр RSA является одним из наиболее распространенных ассиметричных, имеющий применение во многих приложениях.

В основу криптографической системы с открытым ключом RSA положена сложность задачи факторизации произведения двух больших простых чисел. Для шифрования используется операция возведения в степень по модулю большого числа. Для дешифрования (обратной операции) за разумное время необходимо уметь вычислять функцию Эйлера от данного большого числа, для чего необходимо знать разложение числа на простые множители.

В криптографической системе RSA каждый ключ состоит из пары целых чисел. Каждый участник создаёт свой открытый и закрытый ключ самостоятельно. Закрытый ключ каждый из них держит в секрете, а открытые ключи можно сообщать кому угодно или даже публиковать их. Открытый и закрытый ключи каждого участника обмена сообщениями в криптосистеме RSA образуют «согласованную пару» в том смысле, что они являются взаимно обратными.

Алгоритм создания открытого и секретного ключей

RSA-ключи генерируются следующим образом:

1. выбираются (генерируются) два различных случайных простых числа p и q заданного размера (например, 1024 бита каждое). В программе размер

чисел *p* и *q* должен выбирать пользователь. При этом размер чисел должен быть степенью двойки и не должен быть меньше 16 бит.

2. вычисляется их произведение *n=p*q*, которое называется модулем;

3. вычисляется значение функции Эйлера от числа *n*: *φ(n)=(p-1)*(q-1);*

4. выбирается целое число e (1 < e < φ(n)), взаимно простое со значением функции *φ(n)*. Число *e* называется открытой экспонентой. В качестве *e* берут простые числа, из чисел Ферма: 3, 5, 17, 257 или 65537. В программе пользователю должна предоставляться возможность выбора открытой экспоненты;

5. вычисляется целое число *d,* мультипликативно обратное к числу *e* по модулю *φ(n)*, то есть число, удовлетворяющее сравнению:

$$d \cdot e \equiv 1 \pmod{\varphi(n)}$$

Вычисление числа *d* осуществляется при помощи расширенного алгоритма Евклида. Описание этого алгоритма представлено ниже;

6. Таким образом, открытым ключом шифра является пара *(n,e).* Закрытым ключом является набор *(p, q, d).*

Расширенный алгоритм Евклида

Вход: Натуральные числа *x* и *y*: $x \geq y$

Выход: *m=НОД(x, y)*, целые числа *a* и *b* такие, что $ax + by = d$. НОД – наибольший общий делитель.

1. $a_2 = 1, a_1 = 0, b_2 = 0, b_1 = 1.$

2. Пока $y \neq 0$, выполнять

2.1. $q = \left[\frac{x}{y}\right], r = x - qy, a = a_2 - qa_1, b = b_2 - qb_1$. [ ]- целая часть числа.

2.2. $x = y, y = r, a_2 = a_1, a_1 = a, b_2 = b_1, b_1 = b.$

3. $m = x, a = a_2, b = b_2.$

4. Результат: $m, a, b.$

Для алгоритма RSA в расширенном алгоритме Евклида *x = (p-1)*(q-1)*, *y = e*, *b = d*. При выполнении алгоритма Евклида необходимо контролировать получившееся в ходе данного алгоритма число *m*. Необходимо, чтобы было *m=1*. В противном случае необходимо выбрать другое *e*.

Алгоритм генерации простого числа

Для генерации простых чисел *p* и *q* можно воспользоваться сторонними библиотеками или использовать следующий алгоритм:

Вход: Разрядность k искомого простого числа p; параметр $t \geq 1$.

Выход: Число p, простое с вероятностью $1 - \frac{1}{4^t}$

1. Сгенерировать случайное k-битное число $p = b_{k-1}b_{k-2} \dots b_o,\ b_i \in \{0,1\}$.

2. Положить $b_{k-1} = 1, b_0 = 1.$

3. Проверить, что p не делится на простые числа 3,5,7,... (данный шаг можно опустить).

4. Для $i = \overline{1,t}$ выполнить следующие действия

4.1. Выбрать случайное число $a, 2 \leq a \leq p - 2.$

4.2. Проверить число p (переведя его в десятичную систему) тестом Миллера-Рабина для основания a. Если тест выдал, что число p вероятно простое, то вернуться на шаг 4.1 и увеличить счетчик i. В противном случае прекратить цикл и вернуться на шаг 1.

5. Результат: p.

Если для генерации простых чисел p и q Вы используете сторонние библиотеки, то числа также необходимо проверить на простоту при помощи теста Миллера-Рабина, то есть необходимо выполнить пункт 4.

Алгоритм шифрования и расшифрования

Зашифрование производится ПОСИМВОЛЬНО по следующей формуле:

$c = E(m) = m^e \mod n$, где *mod* – операция взятия остатка от деления на число; *c* – шифртекст; *m* – открытый текст.

Приведем пример: допустим мы хотим зашифровать следующий текст “Hello”. Сначала текст переводится в нижний регистр (‘hello’) и затем переводится в числовой формат в соответствии с таблицей ASCII. Получаем следующий результат: 104 101 108 108 111. Затем из полученных чисел вычитается константа 96 для того, чтобы получить порядковый номер буквы в алфавите. Получаем следующий результат: 8 5 12 12 15 (данный шаг необходим, чтобы уменьшить порядок чисел).

Следующий шаг – непосредственно шифрование в соответствии с формулой, представленной выше. Пример:

Предположим, что {p, q} = {17, 19}, {e, n} = {5, 323}, {d, n} = {173, 323}.

Произведем шифрование каждого символа в отдельности:

1. Буква ‘h’. $C_1 = 8^5 \mod 323 = 145$
2. Буква ‘e’. $C_2 = 5^5 \mod 323 = 218$
3. Буква ‘l’. $C_3 = 12^5 \mod 323 = 122$
4. Буква ‘l’. $C_4 = 12^5 \mod 323 = 122$
5. Буква ‘o’. $C_5 = 15^5 \mod 323 = 2$

Таким образом, последовательность 145 218 122 122 2 является почти законченным зашифрованным сообщением. Обращаем внимание, что каждое

число должно иметь разрядность числа *n*, в ином случае при расшифровании возникнут проблемы. В нашем случае, буква ‘о’ зашифровалась в одноразрядное число. Для приведения числа к установленной разрядности, его необходимо дополнить нулями в начале (было – ‘е’ = 2; стало – ‘е’ = 002). В завершении итоговую последовательность необходимо объединить воедино: 145218122122002. Итак, последовательность 145218122122002 является итоговым зашифрованным сообщением.

Расшифрование – обратная процедура и производится по следующей формуле:

$m = D(c) = c^d \mod n$. Расшифрованное сообщение должно быть представлено в виде текста, а не числа!

Хранение ключей и сообщений

Открытый и закрытый ключ должны храниться в разный файлах. Стандарт PKCS1 (RFC-8017) устанавливает правила их хранения

Формат хранения открытого ключа:

```
RSAPublicKey ::= SEQUENCE {
             modulus           INTEGER,  -- n
             publicExponent    INTEGER   -- e
         }
```

Формат хранения закрытого ключа:

```
RSAPrivateKey ::= SEQUENCE {
             modulus           INTEGER,  -- n
             publicExponent    INTEGER,  -- e
             privateExponent   INTEGER,  -- d
             prime1            INTEGER,  -- p
             prime2            INTEGER,  -- q
         }
```

Зашифрованное сообщение должно иметь следующую структуру:

```
EncryptedData :: = SEQUENCE {
            contentType                                 ContentType
            contentEncryptionAlgorithmIdentifier        rsaEncryption
            encryptedContent                            EncryptedContent
          }
```

где contentType – определяет тип контента text, image, и т.д. В нашем случае значение этого поля всегда устанавливается как «text»;

ContentEncryptionAlgorithmIdentifier – идентификатор алгоритма шифрования (для шифра RSA - rsaEncryption);

encryptedContent – зашифрованное сообщение.

Дополнительные требования к реализации

1. При зашифровании и расшифровании ключи, открытый текст и зашифрованный текст должны считываться из файлов.
2. Необходимо производить выбор между зашифрованием, расшифрованием, либо процедурой генерации параметров.
3. Генерация параметров должна записывать данные в новый файл, а не перезаписывать его.
4. Программная реализация должна поддерживать работу с большими числами.
5. Нельзя использовать предусмотренные сторонними библиотеками реализации шифра RSA.
6. Для ускорения работы программы рекомендуется не использовать “встроенную” операцию возведения числа в степень, а реализовать один из алгоритмов быстрого возведения в степень.