Модуль 4. Алгоритмы и структуры данных

Тема 4.13.* Ключи шифрования

Оглавление

4.13. Ключи шифрования

«В повседневной жизни почти для всего существуют неформальные протоколы: заказ товаров по телефону, игра в покер, голосование на выборах. Никто не задумывается об этих протоколах, они вырабатывались в течение длительного времени, все знают, как ими пользоваться, и они работают достаточно хорошо»

Брюс Шнайер

Сегодня все больше людей общаются не лично, а через компьютерную сеть. Честность и безопасность многих протоколов человеческого общения основаны на личном присутствии. Разве Вы доверите незнакомцу деньги, чтобы он купил для Вас автомобиль? Или сядете играть в покер с тем, кто жульничает при сдаче карт? А быть может пошлете свой избирательный бюллетень, не удостоверившись, что он дойдет и дойдет невскрытым? И если человек при личной встрече может определить уровень доверия, то компьютеру нужны определенные правила.

Изучением вопросов шифрования данных занимается наука - криптология, включающая криптографию и криптоанализ.

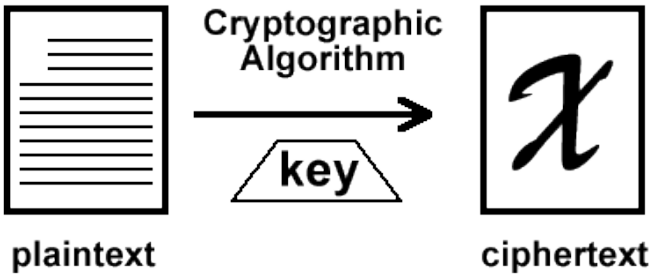
Криптография изучает методы и алгоритмы шифрования данных (правила построения и использования шифров), направленные на то, чтобы сделать эти данные бесполезными для противника.

Методы криптографии также используются для подтверждения подлинности источника данных и контроля целостности данных.

Криптоанализ - это наука о раскрытии исходного текста зашифрованного сообщения без доступа к ключу.

Процесс шифрования информации с точки зрения программиста можно представить следующим образом:

Открытый текст -> алгоритм шифрования (ключ1) -> шифрованное сообщение



А процесс расшифровывания:

Шифрованное сообщение -> алгоритм дешифрирования(ключ2) -> открытый текст.

Здесь:

Открытый текст – информация, которую надо скрыть (в общем случае она совсем не обязательно является текстом, состоящим из символов какого-либо языка; это может быть набор данных, компьютерная программа - что угодно).

Шифрованное сообщение – это информация, скрытая с помощью системы шифрования. Термин «сообщение» используется постольку, поскольку чаще всего шифрование используется для обмена секретной информацией между двумя сторонами, но, в-общем, шифрованное сообщение вовсе не обязано быть куда-либо переданным, оно может быть, например, записано на жесткий диск.

Алгоритм шифрования/дешифрирования – метод, которым пользуются для сокрытия и восстановления информации, а ключ – некоторая специальная, обычно – строго секретная и регулярно изменяющаяся последовательность символов, необходимая для того, чтобы правильно применить алгоритм.

Например, для шифра Цезаря алгоритмом кодирования является замена буквы исходного сообщения на другую букву алфавита, сдвинутую на некоторое число от исходной, а ключом является это число.

Однако может возникнуть вопрос: почему мы говорим о секретности ключа, но не упоминаем о секретности алгоритма? Ведь скрыв реализацию функции “F(открытое сообщение) -> шифрованное сообщение” мы могли-бы не менее надежно защитить секрет. Действительно, ранее считалось, что одним из важнейших способов сохранения тайны является сокрытие метода шифрования, а желательно - даже и наличия шифрованного сообщения (вспомните истории о секретных чернилах, маскировке шпионских донесений под обычную обывательскую корреспонденцию). Что и говорить - даже десятилетие назад утеря во время войны или крушения судна аппаратуры шифрования (которая и реализовывала алгоритм) считалась достаточным основанием для замены всего подобного оборудования на всех объектах. Однако именно этот пример и показывает, что принцип “Безопасность через сокрытие (неясность)” или более привычно - “Security through obscurity” имеет много отрицательных моментов. Утеря техники, разглашение в отрытой прессе принципов работы и даже просто то, что количество алгоритмов принципиально ограничено привело к тому, что в современной криптографии считается, что сами алгоритмы шифрование/дешифровки могут быть открытыми (и это не должно влиять на стойкость криптосистемы), а вот ключи должны быть секретными.

Ключи, используемые для шифрования и дешифрирования информации могут быть как одинаковыми – в этом случае говорят о симметричном шифровании, так и различными – в случае использования асимметричных алгоритмов шифрования.

4.13.1. Симметричный алгоритм шифрования

Симметричный алгоритм шифрования предполагает, что и для прямого преобразования сообщения (из открытого текста в шифрованный), и для обратного - используется один и тот-же ключ, который, очевидно, должен хранится в секрете обеими сторонами. Если две стороны хотят обмениваться зашифрованными сообщениями в безопасном режиме, то обе стороны должны иметь одинаковые симметричные ключи.

Пусть Алиса и Боб хотят безопасно обменяться информацией. Для осуществления обмена должен быть определен протокол действий:

1. Алиса и Боб выбирают систему шифрования.
2. Алиса и Боб выбирают ключ.
3. Алиса шифрует открытый текст своего сообщения с использованием алгоритма шифрования и ключа.

4. Алиса посылает шифрованное сообщение Бобу.
5. Боб дешифрует сообщения с использованием алгоритма шифрования и ключа.

Что может узнать Ева находясь между Алисой и Бобом и подслушивая протокол? Может подслушать передачу сообщения (этап 4). Тогда ей придется подвергнуть сообщения криптоанализу. Так же Ева может послушать этапы 1 и 2. Тогда ей, как и Бобу, станут известны алгоритм и ключ, и когда она перехватит сообщение, сможет его дешифровать.

Кроме этого Ева может попытаться нарушить линию связи так, чтобы сообщение, отправленное Алисой, до Боба не дошло. Или же может перехватить сообщение Алисы, а Бобу отправить свое. А так как Ева знает алгоритм и ключ, то Боб не сможет догадаться, что сообщение отправлено не Алисой.

Шифрование симметричным ключом обычно используется для шифрования большого объема данных, так как этот процесс проходит быстрее, чем при ассиметричном шифровании. Обычно используются алгоритмы DES (Data Encryption Standard – стандарт шифрования данных), 3-DES (тройной DES), RC2, RC4, и AES (Advanced Encryption Standard – современный стандарт шифрования).

XOR-шифрование

Одной из простейших реализаций симметричного алгоритма является XOR-шифрование:

Предположим, что Алиса (A) хочет переслать Бобу (B) конфиденциальное сообщение X, являющееся некоторой последовательность байт. При этом у обоих сторон есть известный только им идентичный ключ Y (последовательность байт длинной не менее длины сообщения. Тогда, Алиса, применяя побайтово операцию XOR (X, Y), получит Z – зашифрованное сообщение, которое сможет переслать Бобу по открытому каналу связи. Для получения исходного сообщения X, Бобу надо только провести операцию XOR над сообщением Z и ключом Y.

В криптографии такой метод (проведение операции «исключающее или» над сообщением и ключом) называется «гаммированием», а персонажи с именами Алиса, Боб и выступающие обычно в качестве злодеев Ева и Чак (Chuck), используются для анализа и иллюстрирования работы алгоритмов.

Математически доказано (К.Шэннон 1948), что, при условии одноразовом использования «статистически хороших» ключей длиной не менее длины входящего сообщения, этот простой шифр (гаммирование) будет абсолютно стойким. Это означает, что противник никогда не сможет извлечь никакой полезной информации из перехвата любого количества зашифрованных сообщений.

Однако использование такой криптосистемы технически довольно трудоемко: для обмена сообщениями между двумя сторонами их надо обеспечить большим

	количеством секретных идентичных наборов длинных последовательностей случайных чисел. Это обходится довольно дорого, поэтому для практических целей обычно используют алгоритмы с ограниченной длиной ключа и неоднократным его использованием. В этом случае для осложнения раскрытия шифра методом частотного анализа в алгоритмах шифрования используются многократно повторяющиеся методы перестановки частей сообщения и гаммирования с ключом. *Так устроен наиболее известный (но уже устаревший) симметричный алгоритм шифрования – DES. На смену ему пришел алгоритм AES (AdvancedEncryptionStandard) — симметричный алгоритм блочного шифрования, принятый в качестве стандарта шифрования правительством США в 2002 г.*

4.13.2. Методы шифрования, основанные на алгоритме DES

Метод DES (Data Encryption Standard) был разработан компанией IBM по заказу правительства США в 1974 году и затем утвержден в качестве стандарта. DES является примером так называемого «блочного шифра», т.е. такого, при применении которого открытый текст разбивается на блоки, состоящие из равного количества бит (обычно 64-256), которые затем обрабатываются раздельно.

В DES существует несколько режимов (фактически – разновидностей алгоритма), наиболее простой из них это - ECB (Electronic Code Book - “электронная кодовая книга”) – режим простой замены. С особенностями реализации прочих режимов, а именно:

- CBC (Cipher Block Chaining) – режим сцепления блоков
- CFB (Cipher Feed Back) – режим обратной связи по шифротексту
- OFB (Output Feed Back) – режим обратной связи по выходу

можно разобраться самостоятельно, зная алгоритм ECB.

Алгоритм ECB

Первым шагом при шифровании сообщения является разбиение его на блоки фиксированной длины. Далее в рассмотрении алгоритма будем считать, что длина блока и длина ключа (k) – 64 бита. Особенностью алгоритма ECB является то, что кодирование полученных блоков осуществляется абсолютно независимо, и, в отличие от прочих режимов DES, результат шифрования предыдущих блоков не используется при обработке всех последующих.

Вторым шагом алгоритма является перестановка отдельных битов блока в соответствии с некоторой (стандартной для данного алгоритма) таблицей перестановки. Цель такой перестановки (как, собственно и перестановок, осуществляемых в методе далее) – скрыть избыточность, регулярность языка и осложнить частотный анализ.

Далее 64-битный блок разбивается на два 32-битных блока L и R, над которыми проводится последовательно 16 раундов преобразований, каждый из которых состоит из следующих действий:

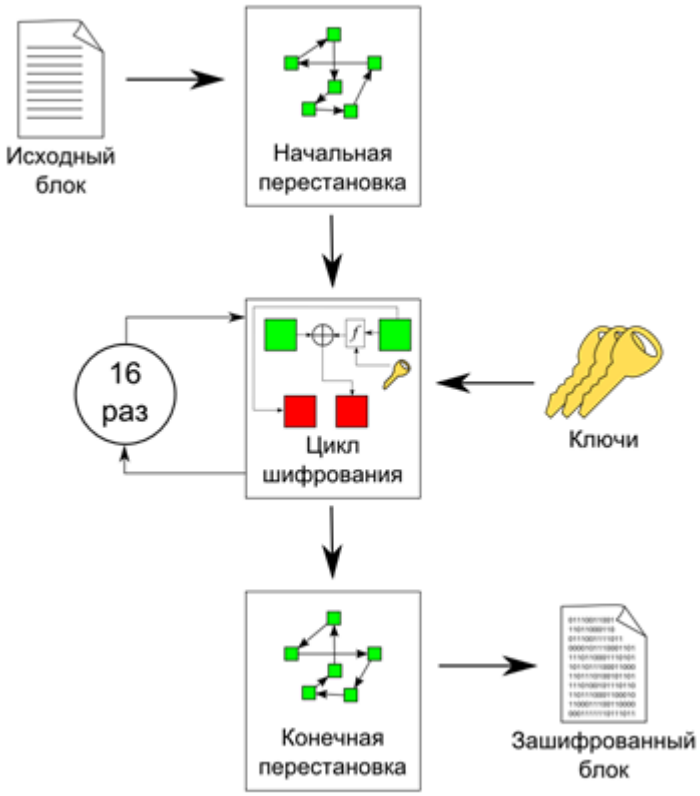
- Перестановка субблоков
- Операция “исключающего или” над одним из субблоков и некоторой функцией f (), которая называется функцией Фейстеля и в качестве аргумента получает ключ

кодирования и преобразуемый субблок: результат равен XOR (блок, f(блок, ключ)).

Сама функция Фейстеля также является последовательностью операций:

- расширения входных данных (удвоения части данных - введение избыточности),
- перестановки отдельных бит,
- XOR результата с ключом,
- повторной перестановки бит.

Эти перестановки функции Фейстеля осуществляются в соответствии со стандартными таблицами алгоритма (называемыми S-блоками).



Создатели метода, отчасти опытным, отчасти - аналитическим путем, подобрали такие таблицы перестановки (и соответствующие функции Фейстеля), что данный алгоритм, с одной стороны, был достаточно быстрым, и - с другой стороны - надежным.

Так, для большинства ключей длиной 64 бита (исключая откровенно слабые, типа 111… или симметричные типа 01FE-01FE-…) вскрытие шифрованного сообщения было практически невозможной задачей вплоть до 90-х годов прошлого века. В 1998 году, используя суперкомпьютер стоимостью 250 тыс. долл., сотрудники RSA Laboratory «взломали» утверждённый правительством США алгоритм шифрования данных (DES) менее чем за три дня.

В дальнейшем развитие компьютеров привело к тому, что взлом подобных шифров методом полного перебора стал возможен за время порядка нескольких дней вычислений. В связи с этим метод DES был усовершенствован – появились методы с увеличенной длиной ключа (до 112 и 168 бит), применением последовательно трех шифрований с разными ключами и еще другие варианты.

Упражнение 4.13.1

В java (и Android) для шифрования применяются классы пакета javax.crypto. Данный пакет входит в состав JDK (и Android SDK). Класс javax.crypto.Cipher реализует все базовые функции всех распространенных криптографических алгоритмов для шифрования данных, включая AES, RSA, DES и нескольких других.

Для того, чтобы создать экземпляр такого класса, используется статистический метод Cipher.getInstance. Данный метод в качестве параметра получает имя криптографического шифрующего алгоритма, например:

Cipher ***cipher*** *= Cipher.getInstance(«DES»);*

После этого нужно настроить экземпляр класса javax.crypto.Cipher с указанием, в каком режиме он должен работать: в режиме дешифрования или шифрования:

cipher*.init(Cipher.ENCRYPT_MODE, key);*

или

cipher.init(Cipher.DECRYPT_MODE, key);

Параметр key – это ключ шифрования криптографического алгоритма DES.

Параметр имеет тип javax.crypto.SecretKey и в общем случае его можно создать следующим образом:

javax.crypto.KeyGenerator.SecretKey key = KeyGenerator.getInstance(«DES»).generateKey()

В алгоритме DES используется симметричное шифрование, а потому для всех операций применяется один и тот же ключ или «master key». Настройка ключа происходит следующим образом.

На основе ключа в виде текста формируется байтовый массив, который может быть использован алгоритмом DES для шифрования данных:

```
DESKeySpec dks = new DESKeySpec(key.getBytes());
```

Класс DESKeySpec предоставляет исходные данные для генерации ключа DES на основе пареметра key - секретного ключа алгоритма DES.

После данной операции на основе байтов ключа происходит создание объекта класса SecretKeyFactory, который представляет фабрику для создания секретных ключей:

```
SecretKeyFactory skf = SecretKeyFactory.getInstance("DES");
```

После этого объект класса SecretKeyFactory затем используется для создания объекта класса SecretKey:

```
SecretKey desKey = skf.generateSecret(dks);
```

который затем передается в объект класса javax.crypto.Cipher, который, в свою очередь, непосредственно производит шифрование или же дешифрование информации.

Класс **`CipherInputStream оборачивает`** входной поток (InputStream) и объект класа **`Cipher так`**, чтобы методы read() возвращали данные, которые были прочитаны из базового входного потока (InputStream) и обработаны шифром.

Например, если объект класа **`Cipher`** был создан для дешифрования информации, то CipherInputStream прочитает данные и попытается расшифровать их, прежде чем вернуть.

```
private String code(String input, String key) throws IOException,
                InvalidKeyException, NoSuchAlgorithmException,
                InvalidKeySpecException, NoSuchPaddingException {
                // пароль для DES шифрования должен быть как минимум 8 символов.
                // Обработайте ошибку так, чтобы пользователь мог использовать
                // и более короткие пароли.
                DESKeySpec dks = new DESKeySpec(key.getBytes());
                SecretKeyFactory skf = SecretKeyFactory.getInstance("DES");
                SecretKey desKey = skf.generateSecret(dks);
                // DES/ECB/PKCS5Padding for SunJCE
                Cipher cipher = Cipher.getInstance("DES");
                cipher.init(Cipher.ENCRYPT_MODE, desKey);
                CipherInputStream     cis     =     new     CipherInputStream(new
ByteArrayInputStream(
                        input.getBytes()), cipher);
                ByteArrayOutputStream output = new ByteArrayOutputStream();
                byte[] buffer = new byte[64];
                int numBytes;
                while ((numBytes = cis.read(buffer)) != -1) {
                output.write(buffer, 0, numBytes);
                }
                // Приводим все байты в кодировку в которой используются только
                // латинские буквы, цифры и некоторые знаки препинания.
                return                 Base64.encodeToString(output.toByteArray(),
Base64.DEFAULT);
        }

        private String decode(String input, String key) throws IOException,
                InvalidKeyException, NoSuchAlgorithmException,
                InvalidKeySpecException, NoSuchPaddingException {
                // Выделите создание секретного ключа в отдельную
                // функцию для уменьшения дублирования кода.
                DESKeySpec dks = new DESKeySpec(key.getBytes());
                SecretKeyFactory skf = SecretKeyFactory.getInstance("DES");
                SecretKey desKey = skf.generateSecret(dks);
                // DES/ECB/PKCS5Padding for SunJCE
                Cipher cipher = Cipher.getInstance("DES");
                cipher.init(Cipher.DECRYPT_MODE, desKey);
                CipherInputStream     cis     =     new     CipherInputStream(new
ByteArrayInputStream(
                        Base64.decode(input, Base64.DEFAULT)), cipher);
                // Выделите копирование содержание потока в
                // строку в отдельный метод.
                ByteArrayOutputStream output = new ByteArrayOutputStream();
                byte[] buffer = new byte[64];
                int numBytes;
                while ((numBytes = cis.read(buffer)) != -1) {
                output.write(buffer, 0, numBytes);
                }
                return output.toString();
        }

        public void processCypher(View context) throws InvalidKeyException,
                NoSuchAlgorithmException, InvalidKeySpecException,
                NoSuchPaddingException, IOException {
                String              input              =              ((EditText)
findViewById(R.id.input_text)).getText()
                        .toString();
```

```
                Log.d("processCypher", "Coding: " + input);
                String              password              =              ((EditText)
findViewById(R.id.password)).getText()
                        .toString();
                Log.d("processCypher", "It is not good idea to log password."
                        + " Any application can get access to it.");
                TextView output = ((TextView) findViewById(R.id.secret_text));
                String secretText = code(input, password);
                output.setText(secretText);
                if (!input.equals(decode(secretText, password))) {
                        // Расшифровка текста приводит к другому результату.
                output.setText("Internal error while coding.");
                }
        }
```

Программа требует ввода пароля как минимум из 8 символов. Исправьте самостоятельно возникающую ошибку при вводе меньшего числа символов.

4.13.3. Асимметричный алгоритм шифрования

Начало асимметричным шифрам было положено в работе «Новые направления в современной криптографии» Уитфилда Диффи и Мартина Хеллмана, опубликованной в 1976 году. Находясь под влиянием работы Ральфа Меркле о распространении открытого ключа, они предложили метод получения секретных ключей, используя открытый канал. Этот метод экспоненциального обмена ключей, который стал известен как обмен ключами Диффи — Хеллмана, был первым опубликованным практичным методом для установления разделения секретного ключа между заверенными пользователями канала. В 2002 году Хеллман предложил называть данный алгоритм «Диффи — Хеллмана — Меркле», признавая вклад Меркле в изобретение криптографии с открытым ключом. Эта же схема была разработана Малькольмом Вильямсоном в 1970-х, но держалась в секрете до 1997 года. Метод Меркле по распространению открытого ключа был изобретён в 1974 и опубликован в 1978 году, его также называют загадкой Меркле.

Как уже было сказано, при использовании асимметричных алгоритмов шифрования для преобразования открытого сообщения в закрытое и обратно используются разные ключи.

Один из ключей используется в качестве открытого. Как правило, специальный (сертификационный) центр публикует открытый ключ в специальном документе (сертификате владельца). Закрытый ключ держится в тайне и никогда никому не открывается. Эти ключи работают в паре: один ключ используется для запуска противоположных функций второго ключа. Так, если открытый ключ используется, чтобы шифровать данные, то расшифровать их можно только закрытым ключом. Если данные шифруются закрытым ключом, то открытый ключ должен это расшифровывать. Такая взаимосвязь позволяет:

- любому пользователю получить открытый ключ по назначению и использовать его для шифрования данных, расшифровать которые может только пользователь, у которого есть закрытый ключ;
- если данные шифруются с использованием закрытого ключа, каждый может расшифровать данные, используя соответствующий открытый ключ.

Пусть Алиса и Боб хотят безопасно обменяться информацией. Для осуществления обмена должен

быть определен протокол действий:

1. Алиса и Боб согласовывают криптосистему с открытыми ключами.
2. Боб посылает Алисе свой открытый ключ.
3. Алиса шифрует свое сообщение и отправляет его Бобу.
4. Боб дешифрует сообщение Алисы с помощью своего закрытого ключа.

Теперь (в отличие от симметричного шифрования) Алисе и Бобу не нужно тайно договариваться о ключе и решать проблему его передачи. Боб публикует свой открытый ключ, например, в газете, а Ева, подслушивающая весь протокол, уже не может расшифровать сообщение, т.к. у нее нет закрытого ключа Боба.

Самый распространенный алгоритм, который используется при шифровании с ассиметричными ключами – RSA (назван в честь разработчиков Rivest, Shamir, Adleman).

Алгоритмы криптосистемы с открытым ключом можно использовать:

- как самостоятельное средство для защиты передаваемой и хранимой информации,
- как средство распределения ключей (обычно с помощью алгоритмов криптосистем с открытым ключом распределяют ключи, малые по объёму, а саму передачу больших информационных потоков осуществляют с помощью других алгоритмов),
- как средство аутентификации пользователей.

Преимущества асимметричных шифров перед симметричными:

- Не нужно предварительно передавать секретный ключ по надёжному каналу.
- Только одной стороне известен ключ шифрования, который нужно держать в секрете (в симметричной криптографии такой ключ известен обеим сторонам и должен держаться в секрете всеми).
- В больших сетях число ключей в асимметричной криптосистеме значительно меньше, чем в симметричной.

4.13.4. Алгоритм RSA

Однако встает вопрос: как создать такой алгоритм и такую пару ключей, чтобы

1) с помощью открытого ключа 1 можно было надежно зашифровать любое сообщение ди
2) расшифровать его можно было только с помощью закрытого ключа 2 и
3) никаким образом нельзя было-бы получить ключ 2 из ключа 1.

Для решения такой задачи используют так называемые “однонаправленные” (one-way) функции, т.е такие, для которых **y=f(x)** вычисляется достаточно быстро и легко, а вот найти x, зная значение y и саму функцию **f()** - невозможно. Математики пока сомневаются, что идеальная однонаправленная функция вообще существует, но для практических целей в качестве такой функции пока вполне можно применять, например, функцию дискретного возведения в степень: **$y = a^x \pmod p$.** Если чуть подробнее, то это возведение в степень **x** достаточно большого целого

числа **a** с последующим получением остатка от целочисленного деления на **p.**

Для дальнейшей работы нам потребуется вспомнить некоторые термины теории чисел.

*Делитель: если положительное целое число **X** делится нацело на целое **N**, то **N** - делитель числа **X**. Так, делителем числа 9 являются 1, 3 и 9. Вообще любое число (кроме 1) всегда имеет минимум два делителя - себя и единицу, однако абсолютное большинство чисел имеет больше делителей.*

Здесь и далее мы рассматриваем только натуральные, то есть целые положительные числа.

Простое число: число Y будет простым, если не имеет иных делителей, чем себя самого и 1. Пример простых чисел: 3, 5, 7, 11.

Взаимно простые числа: два числа будут называться взаимно простыми, если они не имеют общих делителей, кроме 1. Понятно, что два различных простых числа взаимно просты. А вот пример взаимно простых чисел, не являющихся при этом простыми: 9 и 8, 55 и 21.

*И, на всякий случай, напоминаем, запись **X = p (mod N) или X mod N= p** означает, что при делении нацело **X** на **N** остаток будет равен **p.***

Даже без дополнительных условий и математического обоснования понятно, что

1) небольшие изменения **x** в такой функции приведут к существенным изменениям y

2) определить исходное **x** по **y** или очень сложно или вообще невозможно.

Действительно, на текущий момент неизвестен (возможно - вообще не существует) алгоритм быстрого вычисления обратной функции (она называется дискретным логарифмом), особенно при дополнительных ограничениях, накладываемых на значения **a**, **x**, **p**: a должно быть больше **1** и меньше **p**; **1 < x < (p-1)**; а **p** должно быть большим простым числом. Но если нет быстрого алгоритма решения обратной задачи $y = a^x \pmod p$, то, может быть вычисление x по y можно решить простым перебором? Оказывается, что нет - для достаточно больших **a** и **p** это требует слишком больших вычислительных мощностей и слишком большого времени - порядка нескольких лет для современных суперкомпьютеров.

Однако пока не понятно, как пусть даже идеальная однонаправленная функция может помочь в шифровании. Некоторые задачи такая функция действительно успешно решает. Представим, например, задачу по идентификации пользователей при входе в операционную систему компьютера по паролю. Первоначально - при появлении нового пользователя - мы просим его придумать пароль, преобразуем его с помощью однонаправленной функции и в дальнейшем хранить в служебном файле только результат вычислений. При следующей попытке входа пользователь вновь введет пароль, который вновь будет преобразован нашей функций. Если свежеполученный результат совпадёт с данными, хранящимся в файле - пароль верен. То есть фактически, с помощью однонаправленной функции мы сумели создать секретный хеш-код для хранения пароля, причем даже прямой доступ злоумышленника к файлу с хеш-кодами никак не сможет ему помочь, так как получить из результата выполнения функции ее параметр он не сможет (функция однонаправленная), а для входа нужен только пароль - хеш-код не годится.

Такое применение весьма полезно, но задачу безопасного обмена сообщениями между Бобом и Алисой, видимо, описанная выше однонаправленная функция решить не может. И действительно, для шифрования применяется модифицированная однонаправленная функция - так называемая

“однонаправленная функция с потайным ходом” (trapdoor function)..

Однонаправленная функцией с потайным ходом называют такую функцию **y= f_z(x)**, что 1) зная **y**, невозможно (или технически очень трудоемко) получить **x**; 2) зная значение **z**, получить из известного значения **y** значение **x** возможно.

Вариантов однонаправленных функция с потайным ходом, как ни странно, уже найдено довольно много. Ни одна из них не является идеальной, так как, используя неограниченные вычислительные мощности неограниченное время, простым перебором можно найти все решения для любой из этих функций. Однако, используя обычные ограничения на сложность исходных параметров можно добиться того, что задача поиска исходного значения решалась-бы за неприемлемо большое время (годы вычислений). Первым предложенным вариантом данного класса функций является уже упомянутый выше алгоритм RSA.

В качестве однонаправленной функции для преобразования исходного сообщения в зашифрованное используется все ту-же функцию дискретного возведения в степень: **$y = x^e \pmod N$.** Используется здесь эта функция чуть иначе, чем ранее: исходное сообщение здесь само возводится в степень, а не находится в показателе степени, но, остается неясным, как в этой функции создан “потайной вход”? Оказывается, он сформирован с помощью специального подбора параметров **e** и **N**.

Рассмотрим реализацию алгоритма:

1. Возьмем достаточно большие простые числа **p** и **q**.
2. Вычислим произведение этих чисел: **n = p x q**.
3. Для этого произведения вычислим функцию Эйлера **f(n)**: **m = (p – 1) x (q - 1)**
4. Найдем число **d**, взаимно простое с **m**.
5. Найдем число **e** такое, что **e x d = 1 (mod m)**.

Пара **e** и **n** будут открытым ключом алгоритма, а пара **d** и **n** – закрытым ключом.

Процедура шифрирования будет заключаться в

- разбиении исходного сообщения на блоки такой длины, чтобы блоку можно было сопоставить (преобразовать в) число **x < n**
- последовательном применении к блокам функции шифрирования **E(x)** (Encryption): **$y = x^e \pmod n$.**

Для дешифрирования достаточно применить к полученному закодированному сообщению ту-же функцию, но с показателем **d – D(y)** (Decryption): **$x = y^d \pmod n$.**

*Покажем, что алгоритм будет работать, что последовательное выполнение y = **E(x)** и **D(y)** даст нам действительно **x,** то есть: **D(E(x)) = x** или, используя конкретные функции **$x = (x^e \pmod n)^d \pmod n$.** Для доказательства необходимо сослаться на несколько определений и теорем из теории чисел.*

*Функцией Эйлера от числа n **f(n)** называется количество натуральных чисел, меньших, чем **n** и взаимно простых с ним. Понятно, что для простого числа **X** функция Эйлера будет равна **X-1,** ведь все числа, меньшие **X**, будут взаимно просты с ним. Также можно доказать, что функция Эйлера имеет такое свойство, как мультипликативность: если **X = M * N**, то функция Эйлера от **X: f(X) = f(M) * f(N)**;*
*а для простого числа **p** и натурального **n**: **$f(p^n) = p^n – p^{n-1} = p^{n-1}(p-1)$**.*
Малая теорема Ферма (в отличие от большой теоремы Ферма, эта может быть

доказана и даже сравнительно несложно) звучит так: если ***p*** *- простое число, а* ***X*** *- целое число, не делящееся на* ***p****, то* $X^{(p-1)}-1$ *будет нацело делится на* ***p****. Иными словами:* $X^{(p-1)} = 1 (mod\ p)$.

Лемма Эвклида: если ***a, b*** *и* ***d*** *– натуральные числа и* ***d*** *делится нацело на произведение* ***a*** *и* ***b****, то* ***d*** *делится на* ***a*** *или на* ***b****. Следствие леммы: если* $a = M\ (mod\ p)$ *и* $a = M\ (mod\ q)$, *то* $a = M\ (mod\ p*q)$.

А теперь давайте рассмотрим, как именно все это работает в алгоритме RSA. Первыми шагами алгоритма, как описано выше, является подбор двух больших простых числа ***p*** *и* ***q*** *и вычисление их произведения* ***N = p * q****. Для этого произведения, которое в дальнейшем называется "модулем" вычисляем функцию Эйлера* $f(N) = M = f(p)*f(q)= (p-1)*(q-1)$ *(см свойства функции Эйлера для простых чисел и свойство мультипликативности).*
На следующем шаге мы должны выбрать число ***e****, которое должно быть* ***> 1****, но меньше* ***M*** *и взаимно простое с* ***M*** *и дополнительно - найти такое число* ***d****, что для него* $e*d=1(mod\ M)$. *(Отметим, что этот шаг кажется сложным, но для него существует сравнительно простой алгоритм - немного измененный алгоритм Евклида.)*
Теперь докажем, что $x = (x^e\ (mod\ n))^d\ (mod\ N)$.
С начала покажем, что $x^{ed}\ (mod\ p) = x\ (mod\ p)$ *и* $x^{ed}\ (mod\ q) = x\ (mod\ q)$. *Рассмотрим два взаимодополняющих варианта: а)* $x = 0(mod\ p)$ *и б)* $x\ != 1(mod\ p)$

а) Если $x = 0(mod\ p)$, *значит* $x = i * p$, *где* ***i*** *– это некоторое целое число. Тогда* $x^{ed} = (i*p)^{ed} = p* x^{ed-1} * i^{ed}$. *Значит* $x^{ed} = 0(mod\ p)$, *как и* $x = 0(mod\ p)$. *Т.е.* $x^{ed}\ (mod\ p) = x(mod\ p)$.

б) Если $x\ != 0(mod\ p)$. *Это означает, х не делится нацело на р и, значит, согласно малой теореме Ферма:* $x^{(p-1)} = 1(mod\ p)$. *Преобразуем исходное выражение* $x^{ed}\ (mod\ p)$.
Поскольку ***e и d*** *подобраны так, чтобы* $e*d=1(mod\ M)$, *то* $e*d-1 = M*k$, *где* ***k*** *– некоторое целое число, или, иначе говоря, вспомнив, что М – это функция Эйлера для предварительно выбранных нами р и q:* $e*d-1 = M*k = (p-1)*(q-1)*k$. *Значит,* x^{ed} *можно преобразовать* $x^{ed} = x^{ed-1} * x = x^{(p-1)*(q-1)*k} * x = (x^{(p-1)})^{(q-1)*k} * x$ *или по модулю* ***p****:* $x^{ed}\ (mod\ p) = (x^{(p-1)})^{(q-1)*k}(mod\ p)* x\ (mod\ p)$.
Но, как выше было отмечено, мы рассматриваем вариант $x\ != 0(mod\ p)$ *откуда, согласно малой теореме Ферма:* $x^{(p-1)} = 1(mod\ p)$. *То есть выражение* $(x^{(p-1)})^{(q-1)*k}(mod\ p)* x\ (mod\ p)$ *можно преобразовать к* $(1)^{(q-1)*k}(mod\ p)* x\ (mod\ p) = x\ (mod\ p)$.
Таким образом, мы доказали, что $x^{ed}\ (mod\ p) = x\ (mod\ p)$. *Аналогично* $x^{ed}\ (mod\ q) = x\ (mod\ q)$. *Следовательно, в соотвествии со следствием из леммы Эвклида:* $x^{ed}\ (mod\ p*q)$.
Таким образом, мы доказали, что $x = (x^e)^d\ (mod\ n) = (x^d)^e\ (mod\ N)$, *то есть описанный выше алгоритм RSA действительно верный.*

Упражнение 4.13.2

Зашифруем и расшифруем сообщение "DAD" по алгоритму RSA.

1. Выберем p = 11 и q = 13.
2. Определим n = 11 * 13 = 143.
3. Вычислим функцию Эйлера **f(n)**: **m =** (p - 1) * (q - 1) = 10 * 12 = 120.
4. Следовательно, d будет равно, например, 7 (120 mod 7 = 1).
5. Выберем число e по следующей формуле: (e * 7) mod 120 = 1. Значит e будет равно,

например, 103 ((103 * 7) mod 120 = 1).

6. Представим шифруемое сообщение как последовательность чисел в диапозоне от 0 до 32 (незабывайте, что кончается на n-1). Буква A =1, B=2, C=3.

D	A	D
4	1	4

Теперь зашифруем сообщение, используя открытый ключ {e, n} = {103 ,143}

C1 (D) = (4 ^ 103) mod 143 = 108

C2 (A) = (1 ^ 103) mod 143 = 1

C3 (D) = (4 ^ 103) mod 143 = 108

Теперь расшифруем данные, используя закрытый ключ {d, n} = {7, 143}.

M1=(108 ^ 7) mod 143 = 171382426877952 mod 143 = 4 (D)

M2=(1 ^ 7) mod 143 = 1 (A)

M3=(108 ^ 7) mod 143 = 4 (D)

Задание 4.13.1

Самостоятельно добавить в приложение 4.13.1 еще одну кнопку для дешифрования сообщения.

В качестве игрового элемента учащихся можно разбить на пары, один из участников которой составляет и отправляет зашифрованное сообщение, а другой на своем рабочем месте расшифровывает его; затем участники пары меняются ролями. Ученики заинтересованы добавить эту кнопку, чтобы расшифровать полученное сообщение.

Задание 4.13.2

Изменить программу из упражнения таким образом, чтобы использовалось RSA шифрование. Функция кодирования и декодирования может быть объединена в одну, так как процессы шифрования и дешифрования в RSA абсолютно симметричны.

Литература

1. Fermat’s Little Theorem and RSA Algorithm: (https://exploringnumbertheory.wordpress.com/2013/07/08/fermats-little-theorem-and-rsa-algorithm/)
2. What is the relation between RSA & Fermat's little theorem?
3. Алгоритм шифрования RSA
   http://www.e-nigma.ru/stat/rsa/
4. Википедия:
   - RSA - https://ru.wikipedia.org/wiki/RSA
   - Функция Эйлера - https://ru.wikipedia.org/wiki/%D0%A4%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D1%8F_%D0%AD%D0%B9%D0%BB%D0%B5%D1%80%D0%B0
   - Малая теорема Ферма -

https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D0%BB%D0%B0%D1%8F_%D1%82%D0%B5%D0%BE%D1%80%D0%B5%D0%BC%D0%B0_%D0%A4%D0%B5%D1%80%D0%BC%D0%B0

5. Асимметричное шифрование. Как это работает?: http://intsystem.org/1120/asymmetric-encryption-how-it-work/
6. RSA – алгоритм шифрования с открытым ключом: http://www.paveldvlip.ru/algorithms/rsa.html

Благодарности

Компания Samsung Electronics выражает благодарность за участие в подготовке данного материала преподавателям IT ШКОЛЫ SAMSUNG Соколову Михаилу Владимировичу, Покровской Валерии Павловне.