ФЕДЕРАЛЬНОЕ АГЕНТСТВО СВЯЗИ
МОСКОВСКИЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ СВЯЗИ И ИНФОРМАТИКИ

Криптографические методы и средства программной защиты информации

Методические указания
к выполнению лабораторных работ
для студентов и магистрантов по специальности
“...”

Москва 2013

Методические указания являются руководством для выполнения лабораторных работ, проводимых со студентами (специальности …) по курсу «Криптографические методы и средства программной защиты информации».

Методические указания могут быть полезны для инженеров и сотрудников, осваивающих основы криптографии и знакомящихся с основными средствами защиты информации.

Составил Мочалов В. А., Мочалова А.В.
Рецензент

Работа подготовлена на кафедре информационной безопасности и автоматизации

Оглавление

Лабораторная работа №1

Простые симметричные шифры.

Цель работы: Ознакомление с простыми симметричными криптографическими шифрами.

Теоретическая часть [1]:

Симметричные алгоритмы

Среди алгоритмов шифрования, основанных на ключах существует два основных типа: ***симметричные*** и ***ассиметричные***.

Шифрование открытого сообщения ***m*** с помощью некоторой функции шифрования ***E*** на ключе k_1, в результате чего получается шифрованный текст ***c***, принято обозначать так:

$$E_{k_1}(m) = c$$

Дешифрование шифрованного текста ***c*** с помощью некоторой функции дешифрования ***D*** на ключе k_2, в результате чего получается открытое сообщение ***m***, обозначается следующим образом:

$$D_{k_2}(c) = m$$

В случае, когда $k_1 = k_2$ - шифр называется ***симметричным***, в противном же случае – ***ассимметричным***.

Рисунок 1. *Схема криптосистемы, основанной на ключах.*

Симметричные алгоритмы, иногда называемые условными алгоритмами, представляют собой алгоритмы, в которых ключ шифрования может быть рассчитан по ключу дешифрирования и наоборот. В большинстве симметричных алгоритмов ключи шифрования и

дешифрирования одни и те же. Эти алгоритмы, также называемые алгоритмами с секретным ключом или алгоритмами с одним ключом, требуют, чтобы отправитель и получатель согласовали используемый ключ перед началом безопасной передачи сообщений [1].

Симметричные алгоритмы требуют, чтобы отправитель и получатель согласовали используемый ключ перед началом безопасной передачи сообщений. Безопасность симметричного алгоритма определяется ключом, раскрытие ключа означает, что кто угодно сможет шифровать и дешифрировать сообщения. Пока передаваемые сообщения должны быть тайными, ключ должен храниться в секрете.

Подстановочные шифры

Подстановочным шифром называется шифр, который каждый символ открытого текста в шифротексте заменяет другим символом. Получатель инвертирует подстановку шифротекста, восстанавливая открытый текст. В классической криптографии существует четыре типа подстановочных шифров:

- ***Простой подстановочный шифр***, или ***моноалфавитный шифр,*** - это шифр, который каждый символ открытого текста заменяет соответствующим символом шифротекста. Простыми подстановочными шифрами являются криптограммы в газетах.
- ***Однозвучный подстановочный шифр*** похож на простую подстановочную криптосистему за исключением того, что один символ открытого текста отображается на несколько символов шифротекста. Например, "A" может соответствовать 5, 13, 25 или 56, "B" - 7, 19, 31 или 42 и так далее.
- ***Полиграммный подстановочный шифр*** - это шифр, который блоки символов шифрует по группам. Например, "ABA" может соответствовать "RTQ", "ABB" может соответствовать "SLL" и так далее.

- ***Полиалфавитный подстановочный шифр*** состоит из нескольких простых подстановочных шифров. Например, могут быть использованы пять различных простых подстановочных фильтров; каждый символ открытого текста заменяется с использованием одного конкретного шифра.

Знаменитый ***шифр Цезаря***, в котором каждый символ открытого текста заменяется символом, находящегося тремя символами правее по модулю 26 ("A" заменяется на "D," "B" - на "E", ... "W" - на " Z ", "X" - на "A", "Y" - на "B", "Z" - на "C"), представляет собой простой подстановочный фильтр. Он действительно очень прост, так как алфавит шифротекста представляет собой смещенный, а не случайно распределенный алфавит открытого текста.

Ниже приведена программная реализация этого шифра (шифруются только английские буквы).

```
public class CaeserChipher
{
	public static void main(String[] args)
	{
		String inputText    = "hello WORLD";
		byte   inputMas[]   = inputText.getBytes();
		char   outputMas[] = new char[inputMas.length];

		int    key          = 3;

		for(int j = 0; j < inputMas.length; j++)
		{
			char c = (char)inputMas[j];

			char first = 0;

			if(c >= 'a' && c <= 'z')
				first = 'a';
			else if(c >= 'A' && c <= 'Z')
				first = 'A';

			if(first != 0)
				c = (char)( first + (c - first + key + 26) % 26) ;

			outputMas[j] = c;
		}

		String outputText = new String(outputMas);
		System.out.println(outputText);
	}
}
```

ROTI3 - это простая шифровальная программа, обычно поставляемая с системами UNIX. Она также является простым подстановочным шифром. В

этом шифре "A" заменяется на "N," "B" - на "O" и так далее. Каждая буква смещается на 13 мест. Шифрование файла программой ROTI3 дважды восстанавливает первоначальный файл.

P = ROT13 (ROT13 (P))

Перестановочные шифры

В ***перестановочном шифре*** меняется не открытый текст, а порядок символов. В простом *столбцовом перестановочном шифре* открытый текст пишется горизонтально на разграфленном листе бумаги фиксированной ширины, а шифротекст считывается по вертикали (см. Рисунок 2). Дешифрирование представляет собой запись шифротекста вертикально на листе разграфленной бумаги фиксированной ширины и затем считывание открытого текста горизонтально [1].

Открытый текст: столбцевойперестановочныйшифр

Шифрованный текст: серойтвевшоосоилйтчфбпанрцены

с	т	о	л	б	ц
е	в	о	й	п	е
р	е	с	т	а	н
о	в	о	ч	н	ы
й	ш	и	ф	р	

Рисунок 2. *Столбцевой перестановочный шифр.*

Так как символы шифротекста те же, что и в открытом тексте, частотный анализ шифротекста покажет, что каждая буква встречается приблизительно с той же частотой, что и обычно [1].

Ниже приведена программная реализация столбцевого перестановочного шифра.

```
public class Stolbcevoy
{
	public static void main(String[] args)
	{
		String sInput       = "столбцевойперестановочныйшифр";
		int    columns      = 6;
		byte   inputMas[]   = sInput.getBytes();
		byte   encryptMas[] = new byte[sInput.length()];

		int cur   = 0;
```

```
            int index = 0;

            for(int i = 0; i < inputMas.length; i++)
            {
                  encryptMas[i] = inputMas[index];

                  index += columns;

                  if(index >= inputMas.length)
                  {
                        cur++;
                        index = cur;
                  }
            }
            System.out.print(new String(encryptMas));
      }
}
```

Так как символы шифротекста те же, что и в открытом тексте, частотный анализ шифротекста покажет, что каждая буква встречается приблизительно с той же частотой, что и обычно. Это даст криптоаналитику возможность применить различные методы, определяя правильный порядок символов для получения открытого текста. Применение к шифротексту второго перестановочного фильтра значительно повысит безопасность. Существуют и еще более сложные перестановочные фильтры, но компьютеры могут раскрыть почти все из них.

Задание 1: Придумайте свой ***перестановочный шифр***, реализуйте его на любом языке программирования, зашифруйте с помощью вашего шифра произвольное сообщение, после чего дешифруйте его. Опишите достоинства и недостатки шифра.

Задание 2: Придумайте свой ***простой подстановочный шифр (моноалфавитный шифр)***, реализуйте его на любом языке программирования зашифруйте с помощью вашего шифра произвольное сообщение, после чего дешифруйте его.

Задание 3: Придумайте свой ***однозвучный подстановочный шифр***, реализуйте его на любом языке программирования, зашифруйте с помощью вашего шифра произвольное сообщение, после чего дешифруйте его. Опишите достоинства и недостатки шифра.

Задание 4: Придумайте ***полиграммный подстановочный шифр***, реализуйте его на любом языке программирования, зашифруйте с помощью вашего

шифра произвольное сообщение, после чего дешифруйте его. Опишите достоинства и недостатки шифра.

Задание 5: Придумайте ***полиалфавитный подстановочный шифр,*** реализуйте его на любом языке программирования, зашифруйте с помощью вашего шифра произвольное сообщение, после чего дешифруйте его. Опишите достоинства и недостатки шифра.

Задание 6: Известно, что сообщение "Pbatenghyngvbaf! Vg'f n Pnrfne pvcure!" было зашифровано шифром Цезаря, но ключ *k* неизвестен. Напишите вспомогательную программу, которая поможет вам в расшифровке этого сообщения.

Задание 7: Реализуйте собственный аналог функции **ROT13** его на любом языке программирования, зашифруйте с помощью вашего шифра произвольное сообщение, после чего дешифруйте его. Опишите достоинства и недостатки шифра.

Контрольные вопросы:

1. Какое шифрование называется симметричным?
2. Сколько ключей используется в алгоритмах симметричного шифрования?
3. В чем опасность шифрования симметричными алгоритмами?
4. Опишите основную идею подстановочных шифров.
5. Перечислите все известные Вам типы перестановочных шифров.
6. Опишите основную идею перестановочных шифров.
7. Назовите достоинства и недостатки перестановочных и подстановочных шифров.

Литература:

1. Шнайер, Б. Прикладная криптография. Протоколы, алгоритмы, исходные тексты на языке Си / Б. Шнайер – М.: ТРИУМФ, 2002. – 816с.

Лабораторная работа №2

XOR-шифрование. Одноразовый блокнот.

Тема: «XOR-шифрование. Одноразовые блокноты»

Цель работы: Научиться применять операцию XOR в шифровании, научиться шифровать методом «одноразового блокнота».

Теоретическая часть [1]:

XOR

XOR представляет собой операцию "исключающее или": '^' в языке C. Это обычная операция над битами:

$$0 \oplus 0 = 0$$

$$0 \oplus 1 = 1$$

$$1 \oplus 0 = 1$$

$$1 \oplus 1 = 0$$

Также заметим, что:

$$a \oplus a = 0$$

$$a \oplus b \oplus b = a$$

Открытый текст подвергается операции "исключающее или" вместе с ключевым текстом для получения шифротекста. Так как повторное применение операции XOR восстанавливает оригинал для шифрования и дешифрирования используется одна и та же программа:

$$P + K = C$$

$$C + K = P$$

Ниже приведена программная реализация XOR-шифрования и расшифровки:

```
import java.util.Arrays;

public class Xor
{
	public static void main(String[] args)
	{
		byte openText[]   = {'h', 'e', 'l', 'l', 'o', 1 , 2, 3, 4, 5};
```

```
            byte key[]        = {5, 9, 12};
            byte encryptMas[] = new byte[openText.length];
            byte decryptMas[] = new byte[openText.length];

            for(int i = 0; i < openText.length; i++)
                  encryptMas[i] = (byte)(openText[i] ^ key[i % 3]);

            for(int i = 0; i < openText.length; i++)
                  decryptMas[i] = (byte)(encryptMas[i] ^ key[i % 3]);

            boolean bOK = Arrays.equals(openText, decryptMas);

            if(bOK)
                  System.out.print("Сообщение восстановлено");
      }
}
```

Одноразовый блокнот

Метод шифрования, называемый ***одноразовым блокнотом*** был изобретен в 1917 году Мэйджором Джозефом Моборном (Major Joseph Mauborgne) и Гилбертом Вернамом (Gilbert Vernam) из AT&T. В классическом понимании одноразовый блокнот является большой неповторяющейся последовательностью символов ключа, распределенных случайным образом, написанных на кусочках бумаги и приклеенных к листу блокнота. Первоначально это была одноразовая лента для телетайпов. Отправитель использовал каждый символ ключа блокнота для шифрования только одного символа открытого текста. Шифрование представляет собой сложение по модулю 26 символа открытого текста и символа ключа из одноразового блокнота.

Каждый символ ключа используется только единожды и для единственного сообщения. Отправитель шифрует сообщения и уничтожает использованные страницы блокнота или использованную часть ленты. Получатель, в свою очередь, используя точно такой же блокнот, дешифрирует каждый символ шифротекста. Расшифровав сообщение, получатель уничтожает соответствующие страницы блокнота или часть ленты. Новое сообщение - новые символы ключа. Например, если сообщением является:

ONETIMEPAD

а ключевая последовательность в блокноте:

TBFRGFARFM

то шифротекст будет выглядеть как:

IPKLPSFHGQ

так как

$Q + T \bmod 26 = I$

$N + B \bmod 26 = P$

$E + F \bmod 26 = K$

и т.д.

В предположении, что злоумышленник не сможет получить доступ к одноразовому блокноту, использованному для шифрования сообщения, эта схема совершенно безопасна. Данное шифрованное сообщение на вид соответствует любому открытому сообщению того же размера.

Так как все ключевые последовательности совершенно одинаковы (помните, символы ключа генерируются случайным образом), у противника отсутствует информация, позволяющая подвергнуть шифротекст криптоанализу. Кусочек шифротекста может быть похож на:

POYYAEAAZX

что дешифрируется как:

SALMONEGGS

или на:

BXEGBMTMXM

что дешифрируется как:

GREENFLUID

Так как все открытые тексты равновероятны, у криптоаналитика нет возможности определить, какой из открытых текстов является правильным. Случайная ключевая последовательность, сложенная с неслучайным открытым текстом, дает совершенно случайный шифротекст, и никакие вычислительные мощности не смогут это изменить. Символы ключа должны генерироваться случайным образом.

Ключевую последовательность никогда нельзя использовать второй раз. Даже если вы используете блокнот размером в несколько гигабайт, то если криптоаналитик получит несколько текстов с перекрывающимися ключами, он сможет восстановить открытый текст. Он сдвинет каждую пару шифротекстов относительно друг друга и подсчитает число совпадений в каждой позиции. Если шифротексты смещены правильно, соотношение совпадений резко возрастет - точное значение зависит от языка открытого текста. С этой точки зрения криптоанализ не представляет труда. Это похоже на показатель совпадений, но сравниваются два различных "периода". Не используйте ключевую последовательность повторно.

Ниже приведена программная реализация, демонстрирующая шифрование и дешифрование сообщения с помощью одноразового блокнота.

```
import java.util.Random;

public class Bloknot
{
	public static void main(String[] args)
	{
		if(args.length == 0)
			return;

		byte   bloknotMas[] = new byte[1000];

		//// вместо инициализации блокнота
		Random rand         = new Random();
		rand.nextBytes(bloknotMas);
		////

		byte ret[][] = new byte[args.length][];

		int iCurIndex = 0;
		for(int j = 0; j < args.length; j++)
		{
			String  str           = args[j];
			byte   mas[]          = str.getBytes();

			ret[j] = new byte[mas.length];

			int k = 0;

			for( int i = 0; i < mas.length; i++, k++)
			{
				if(iCurIndex + i >= bloknotMas.length)
				{
				// вместо смены страницы блокнота и уничтожения старой
					rand.nextBytes(bloknotMas);
					iCurIndex = 0;
					k         = 0;
				}
```

```
                    ret[j][i] = (byte)(mas[i] ^ bloknotMas[iCurIndex+k]);
                }

                iCurIndex += k;
            }
        }
}
```

Идея одноразового блокнота легко расширяется на двоичные данные. Вместо одноразового блокнота, состоящего из букв, используется одноразовый блокнот из битов. Вместо сложения открытого текста с ключом одноразового блокнота используйте XOR. Для дешифрирования примените XOR к шифротексту с тем же одноразовым блокнотом. Все остальное не меняется, и безопасность остается такой же совершенной.

Задание 1: Реализуйте на любом языке программирования XOR-шифрование, зашифруйте с помощью вашего шифра произвольное сообщение, после чего дешифруйте его.

Задание 2: Реализуйте на любом языке программирования шифрование методом одноразового блокнота, зашифруйте с помощью вашего шифра сообщение на ***русском*** языке, после чего дешифруйте его.

Задание 3: Реализуйте на любом языке программирования шифрование методом одноразового блокнота, зашифруйте с помощью вашего шифра сообщение, состоящее из двоичных данных (используя XOR), после чего дешифруйте его.

Контрольные вопросы:

1. В чем заключается смысл шифрования одноразовым блокнотом?
2. Что такое XOR-шифрование? Где его применяют?
3. Почему шифрование методом одноразового блокнота считается идеальным шифрованием.
4. Какие достоинства и недостатки шифрования методом одноразового блокнота вы можете выделить?

5. Каким образом можно получить данные, зашифрованные с помощью одноразового блокнота?
6. Каким образом можно получить данные, зашифрованные XOR?

Литература:

1. Шнайер, Б. Прикладная криптография. Протоколы, алгоритмы, исходные тексты на языке Си / Б. Шнайер – М.: ТРИУМФ, 2002. – 816 с.

Лабораторная работа №3

Криптографическая программная библиотека Bouncy Castle

В настоящее время существует множество криптографических библиотек. В лабораторной будет рассматриваться использование библиотеки с открытым исходным кодом *Bouncy Castle* (скачать библиотеку можно по адресу http://www.bouncycastle.org/). Этой библиотеке посвящена целая книга [1]. Библиотека *Bouncy Castle* работает совместно с *Java Cryptography Extension (JCE)* и *Java Cryptography Architecture (JCA).*

Для динамического подключения провайдера *Bouncy Castle* требуется:

- добавить в *Java*-проект библиотеки *Bouncy Castle* (на текущий момент это bcprov-jdk16-143.jar, bctsp-jdk16-143.jar, bcmail-jdk16-143.jar, bcpg-jdk16-143.jar и bctest-jdk16-143.jar),
- импортировать класс BouncyCastleProvider (import org.bouncycastle.jce.provider.BouncyCastleProvider;),
- подключить провайдер Security.addProvider(new BouncyCastleProvider()).

Для статического подключения провайдера *Bouncy Castle* требуется:

- скопировать библиотеки *Bouncy Castle* в папку $JAVA_HOME/jre/lib/ext,

- в файл $JAVA_HOME/jre/lib/security/java.security добавить *BouncyCastle* в список провайдеров security.provider.N=org.bouncycastle.jce.provider.BouncyCastleProvider , указав N. Например, security.provider.10=org.bouncycastle.jce.provider.BouncyCastleProvider ,

- при наличии предупреждений в среде разработке *Eclipse* выберите ваш проект и затем выберете в меню Project -> Properties -> Java Compiler -> Errors/Warnings -> Deprecated and restricted API установите всем полям (combo box) значение Warning.

Для просмотра установленных криптографических провайдеров может быть выполнен следующий программный код:

```
import java.security.Provider;
import java.security.Security;
import org.bouncycastle.jce.provider.BouncyCastleProvider;

public class InstalledProviders
{
      static
      {
            Security.addProvider(new BouncyCastleProvider());
      }

    public static void main(String[]     args)
    {
        Provider[]      providers = Security.getProviders();

        for (Provider provider : providers)
        {
            System.out.println("Название: " + provider.getName() + "; Версия:
" + provider.getVersion());
        }
    }
}
```

Результат работы программы:

Название: SUN; Версия: 1.6
Название: SunRsaSign; Версия: 1.7
Название: SunJSSE; Версия: 1.6
Название: SunJCE; Версия: 1.7

Название: SunJGSS; Версия: 1.0
Название: SunSASL; Версия: 1.5
Название: XMLDSig; Версия: 1.0
Название: SunPCSC; Версия: 1.6
Название: SunPKCS11-NSS; Версия: 1.7
Название: BC; Версия: 1.47

Для просмотра возможностей провайдера *Bouncy Castle* может быть выполнен следующий программный код:

```
import java.security.Provider;
import java.security.Security;
import java.util.ArrayList;
import java.util.Hashtable;
import java.util.Iterator;
import java.util.Map.Entry;

public class CapabilitiesOfBC
{
    public static void main(String[] args)
    {
        Provider  provider = Security.getProvider("BC");
        Iterator  it       = provider.keySet().iterator();
        Hashtable<String, ArrayList<String>> htTypeNames = new
Hashtable<String, ArrayList<String>>();

        while (it.hasNext())
        {
            String     entry = (String)it.next();

            if (entry.startsWith("Alg.Alias."))
            {
                entry = entry.substring("Alg.Alias.".length());
            }

            String  typeClass = entry.substring(0, entry.indexOf('.'));
            String  name      = entry.substring(typeClass.length() + 1);

            if(htTypeNames.containsKey(typeClass) == false)
                  htTypeNames.put(typeClass, new ArrayList<String>());

            htTypeNames.get(typeClass).add(name);
        }

        for(Entry<String, ArrayList<String>> typeArrayEntry :
htTypeNames.entrySet())
        {
            String            type    = typeArrayEntry.getKey();
            ArrayList<String> arNames = typeArrayEntry.getValue();

            System.out.println(type + ":");

            for(String name : arNames)
                  System.out.println(name);

            System.out.println("--------------------");
        }
```

```
    }
}
```

Ниже приведен программный код шифрования различными известными алгоритмами с использованием библиотеки *Bouncy Castle*.

ГОСТ 28147-89, AES, DES, BLOWFISH

```
import java.security.InvalidKeyException;
import java.security.NoSuchAlgorithmException;
import javax.crypto.KeyGenerator;
import javax.crypto.BadPaddingException;
import javax.crypto.Cipher;
import javax.crypto.IllegalBlockSizeException;
import javax.crypto.SecretKey;
import org.bouncycastle.util.encoders.Hex;

public class SymmetricTest
{
      public static SecretKey generateSymmetricKey(String algo) throws
NoSuchAlgorithmException
      {
        KeyGenerator kg  = KeyGenerator.getInstance(algo);
        return kg.generateKey();
      }

      public static byte [] simpleSymmetricEncrype(Cipher cipher, SecretKey
key, byte[] inputMas) throws InvalidKeyException, IllegalBlockSizeException,
BadPaddingException
      {
            cipher.init(Cipher.ENCRYPT_MODE, key);
            return cipher.doFinal(inputMas);
      }

      public static byte [] simpleSymmetricDecrypt(Cipher cipher, SecretKey
key, byte[] encryptMas) throws InvalidKeyException,
IllegalBlockSizeException, BadPaddingException
      {
            cipher.init(Cipher.DECRYPT_MODE, key);
            return cipher.doFinal(encryptMas);
      }

      public static void main(String[] args)  throws Exception
      {
            byte   inputMessage[]  = "Привет Мир!".getBytes();
            String masAlgos[]      = {"GOST-28147", "AES", "DES", "BLOWFISH"};

            for(String algo : masAlgos)
            {
                  Cipher    cipher   = Cipher.getInstance(algo, "BC");
                  SecretKey key      = generateSymmetricKey(algo);

                  byte      encMas[] = simpleSymmetricEncrype(cipher, key,
inputMessage);
                  byte      decMas[] = simpleSymmetricDecrypt(cipher, key,
encMas);

                  System.out.println("Алгоритм: " + algo);
                  System.out.println("Секретный ключ: 0x" + new
String(Hex.encode(key.getEncoded())));
```

```
                System.out.println("Зашифрованное сообщение - 0x" + new
String(Hex.encode(encMas)));
                System.out.println("Расшифрованное сообщение - " + new
String(decMas));

                System.out.println("----------------------");
            }
        }
}
```

Алгоритм: GOST-28147

Секретный ключ: 0xff63a6d62bf332f882e5239228d7a93e472968c7129a5c0a85c89b14a6821a58

Зашифрованное сообщение - 0x0c72ea4efc06c35f4e699ab4a88050ac

Расшифрованное сообщение - Привет Мир!

Алгоритм: AES

Секретный ключ: 0x1563b74f769170a1ce826d7674951649

Зашифрованное сообщение - 0x34fe3164187feda4ef32a583cd7bc17c

Расшифрованное сообщение - Привет Мир!

Алгоритм: DES

Секретный ключ: 0x29cdbab0e998b392

Зашифрованное сообщение - 0x2e16d95f6566c6ccbb211f0d6fc8637f

Расшифрованное сообщение - Привет Мир!

Алгоритм: BLOWFISH

Секретный ключ: 0x5f5a95080e3fba0fe4c34fb4ec9e70a3

Зашифрованное сообщение - 0x745f3d8b20c21938aa597d681ae292d1

Расшифрованное сообщение - Привет Мир!

Задание: С использованием библиотеки *Bouncy Castle* зашифруйте файл с помощью поддерживаемого библиотекой симметричного алгоритма шифрования, согласованного с преподавателем.

Контрольные вопросы

1. Для чего предназначена библиотека *Bouncy Castle* ?
2. Что такое *Java Cryptography Extension* ?
3. Что такое *Java Cryptography Architecture* ?
4. Какими функциональными возможностями обладает библиотека *Bouncy Castle* ?
5. Поддерживает ли библиотека *Bouncy Castle* работу с ГОСТ Р 34.10-2012 и ГОСТ Р 34.11-2012 ?

Литература

1. David Hook. Beginning Cryptography with Java, 2005, 480 С.

Лабораторная работа №4

Криптографические алгоритмы с открытыми ключами. RSA.

Цель работы: создать учебный вариант криптографической системы с открытым ключом на алгоритме RSA.

Теоретическая часть [1, 2]:

Самой первой криптографической системой с открытым ключем, из тех что были предложены в открытой литературе вообще, является криптосистема Ривеста, Шамира и Эдлемана [1]. Она стала известна под названием RSA [2].

Криптосистема RSA стала первой системой, пригодной и для шифрования, и для цифровой подписи. Алгоритм используется в большом числе криптографических приложений, включая PGP, S/MIME, TLS/SSL, IPSEC/IKE и других.

Формирование системы RSA

1. Выбираем два различных простых числа p и q.
2. Вычисляем $n = pq$, $\varphi(n)=(p-1)(q-1)$.
3. Выбираем число e, взаимно простое с $\varphi(n)$.

4. Вычисляем число d из уравнения $de \equiv 1 \bmod \varphi(n)$.
5. Определяются открытые ключи e и n.
6. Определяются закрытые ключи d, p, q и $\varphi(n)$.

Алгоритм шифрования

1. Дан текст сообщения M.
2. Шифрованный текст C вычисляется по формуле $C=E_K(M)=M^e \bmod n$.

Алгоритм дешифрования

1. Дан шифрованный текст C.
2. Текст сообщения M вычисляется по формуле

   $M=D_K(C)=C^d \bmod n \equiv (M^e)^d \bmod n \equiv M$

Ниже приведен программный код, реализующий криптографическую систему RSA, с использованием криптографической программной библиотеки с открытыми исходными кодами (http://www.bouncycastle.org/).

```
import java.security.Key;
import java.security.KeyPair;
import java.security.KeyPairGenerator;
import java.security.SecureRandom;
import java.security.Security;
import org.bouncycastle.jce.provider.BouncyCastleProvider;
import org.bouncycastle.util.encoders.Hex;
import javax.crypto.Cipher;

public class RSATest
{
      static
      {
            Security.addProvider(new BouncyCastleProvider());
      }

      public static void main(String[] args)  throws Exception
      {
        Cipher           cipher    = Cipher.getInstance("RSA", "BC");
        SecureRandom     random    = new SecureRandom();
        KeyPairGenerator generator = KeyPairGenerator.getInstance("RSA",
"BC");

        generator.initialize(512, random);

        KeyPair          pair    = generator.generateKeyPair();
        Key              pubKey  = pair.getPublic();
        Key              privKey = pair.getPrivate();
        String           sInput  = "Привет МИР!";

        cipher.init(Cipher.ENCRYPT_MODE, pubKey, random);
        byte[] cipherText = cipher.doFinal(sInput.getBytes());
        System.out.println("Зашифрованное сообщение - 0x" + new
String(Hex.encode(cipherText)));
```

```
        cipher.init(Cipher.DECRYPT_MODE, privKey);
        byte[] plainText = cipher.doFinal(cipherText);
        System.out.println("Расшифрованное сообщение - " + new
String(plainText));
      }
}
```

Результат работы программы:

Зашифрованное сообщение -

0x3d0e3fcdd4e101af8e4adcfc5bc38126db150b2961f38693fabace2d5de41ba506f7c697ad7c860de0fe0471b614f5fd1f036a6e0a81864a77312b3e2e30510f

Расшифрованное сообщение - Привет МИР!

Задание: создать программную реализацию системы шифрования RSA без использования криптографических программных библиотек.

Содержание отчета:

1. Титульный лист
2. Задание
3. Описание работы системы RSA
4. Пошаговый алгоритм работы реализуемой системы
5. Набор входных и выходных данные программы, позволяющие составить полное впечатление о ее работе.
6. Выводы о принципах работы системы RSA, ее криптостойкости.

Контрольные вопросы

1. Дать определение односторонней функции.
2. Дать определение односторонней функции с секретом.
3. На какой базе обычно создаются криптографические системы с открытыми ключами?
4. Перечислить число параметров в криптографической системе RSA.
5. Перечислить секретные параметры системы RSA.
6. Перечислить открытые параметры системы RSA.
7. На каком математическом результате базируется система RSA.
8. Какими свойствами должны удовлетворять параметры p и q системы RSA.

9. Какими свойствами должны удовлетворять параметры e и d системы RSA.
10. На какой достаточно трудной задаче из теории чисел базируется криптографическая система RSA.
11. Опишите схему шифрования текста с использованием алгоритма RSA.
12. Опишите схему дешифрования текста с использованием алгоритма RSA.
13. Опишите схему формирование цифровой подписи с применение алгоритма RSA.
14. Сформулировать теорему Ферма.
15. Сформулировать теорему Эйлера.

Литература

1. Rivest, R. L., Shamir, A., Adleman, L. M., "A method for obtaining digital signatures and public-key cryptosystems", Communications of the ACM, vol. 21, 1978, pp. 120 - 126.

2. Ж. Брассар. Современная криптология. Руководство. М.: Полимед, 1999.

Лабораторная работа №5

Криптографические хэш-функции.

Цель работы: Изучить существующие алгоритмы вычисления хэшей сообщений и написать программу, реализующую заданный в варианте алгоритм хэширования.

Теоретическая часть [1]:

Функция хеширования (хэш-функция) H представляет собой отображение, на вход которого подается сообщение переменной длины m, а выходом является строка фиксированной длины H(m). В общем случае H(m) будет гораздо меньше, чем m, например, H(m) может быть 128 или 256 бит, тогда как m может быть размером в мегабайт или более [1].

Для того, чтобы функция хэширования могла должным образом быть использована в процессе аутентификации, функция хэширования H должна обладать следующими свойствами [1]:

1. H можеть быть применена к аргументу любого размера;
2. Выходное значение H имеет фиксированный размер;
3. H(x) достаточно просто вычислить для любого x.
4. Для любого y с вычислительной точки зрения невозможно найти H(x) = y.
5. Для любого фиксированного x с вычислительной точки зрения невозможно найти x' != x, такое что H(x) = H(x').

Ниже приведен программный код, демонстрирующий как легко с помощью библиотеки *Bouncy Castle* можно получить дайджест сообщения «Привет мир!», захэшированного MD-5 алгоритмом.

```
import java.math.BigInteger;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

public class Hash
{
	public static void main(String[] args) throws NoSuchAlgorithmException
	{
		String sInput	= "Привет, мир!";
		MessageDigest hash = MessageDigest.getInstance("MD5");
		byte[] digest = hash.digest(sInput.getBytes());
		BigInteger biHash3411Out = new BigInteger(digest);
		System.out.println( biHash3411Out.toString(16));
	}
}
```

Задание: реализуйте следующие алгоритмы хэширования, используя криптографические программные библиотеки с открытыми исходными кодами (http://www.bouncycastle.org/ - Java библиотека; http://www.cryptopp.com/ - C++ библиотека):

1. ГОСТ Р 34.11-94.
2. Алгоритм RIPEMD–320.
3. Алгоритм MD5.

4. Алгоритм SHA–256.
5. Алгоритм RIPEMD–128.
6. Алгоритм RIPEMD–160.
7. Алгоритм SHA–512.
8. Алгоритм SHA–384.
9. Алгоритм RIPEMD–256.
10. Алгоритм SHA–224.

Захэшируйте:

1. 3 одинаковых сообщения;
2. 3 сообщения, отличающихся одним символом;
3. сообщение, размер которого не превышает 1 Мб
4. сообщение, размер которого больше 1Мб, но меньше 3 МБ;
5. сообщение, размер которого больше 10Мб.

Сравните хэш-функции 9-ти вышеописанных сообщений, зашифрованных 10-ю различными алгоритмами хэширования.

Контрольные вопросы:

1. Что такое хэш-функция?
2. Когда она является криптографически стойкой?
3. Что такое лавинный эффект?
4. Какими свойствами должна обладать хэш-функция?
5. Где используются хэш-функции?
6. Что такое однонаправленные функции? Где они применяются? Почему так важно свойство однонаправленности функции в хэшировании?

Литература:

1. Баричев С.Г, Серов Р.Е. Основы современной криптографии: Учебное пособие. - М.: Горячая линия - Телеком, 2002.

Лабораторная работа №6

Электронная цифровая подпись.

Цель работы: Ознакомиться со схемами цифровой подписи и получить навыки создания и проверки подлинности ЭЦП.

Теоретическая часть [1, 2]:

Электронная подпись предназначена для идентификации лица, подписавшего электронный документ, и является полноценной заменой (аналогом) собственноручной подписи в случаях, предусмотренных законом [1].

Существует несколько схем построения цифровой подписи:

- На основе алгоритмов *симметричного шифрования*. Данная схема предусматривает наличие в системе третьего лица — арбитра, пользующегося доверием обеих сторон. Авторизацией документа является сам факт зашифрования его секретным ключом и передача его арбитру.[2]
- На основе алгоритмов *асимметричного шифрования*. На данный момент такие схемы ЭП наиболее распространены и находят широкое применение.

Кроме этого, существуют другие разновидности цифровых подписей (групповая подпись, неоспоримая подпись, доверенная подпись), которые являются модификациями описанных выше схем.[2]

В настоящее время цифровые подписи все чаще встречаются в повседневной жизни: для идентификации пользователей, подписывающие документы в налоговой инспекции, в банках, при заключении различных договоров в частных фирмах и т. д.

Задание: реализуйте следующие алгоритмы ЭЦП, используя криптографические программные библиотеки с открытыми исходными кодами (: http://www.bouncycastle.org/ - Java библиотека; http://www.cryptopp.com/ - C++ библиотека):

- подпись RSA,
- подпись Эль-Гамаля.

Проведите эксперименты, моделирующие проверку авторства истинности документа с помощью ЭЦП. Опишите эксперименты в отчете и прокомментируйте результаты.

Контрольные вопросы:

1. Для чего нужна цифровая подпись?
2. Назовите основные свойства цифровой подписи.
3. Какие схемы цифровой подписи существуют? Какая схема самая распространенная?
4. Как осуществляется подпись RSA? В чем отличие подписи RSA от шифра RSA?
5. Как осуществляется подпись Эль-Гамаля?
6. Как осуществляется проверка на подлинность подписи Эль-Гамаля?

Литература:

1. Гражданский кодекс Российской Федерации, часть 1, глава 9, статья 160.
2. Федеральный закон Российской Федерации от 6 апреля 2011 г. N 63-ФЗ «Об электронной подписи».

Лабораторная работа №7

Программная реализация криптографических протоколов.

Цель работы: обрести навыки работы с различными криптографическими протоколами внутри одной криптографической системы

Задание: реализовать криптографическую систему, передающую сообщение от А к В по схеме, представленной на рисунке 7.1. и 7.2., заменив шифрование, хеш и подпись на алгоритмы, указанные в вариантах.

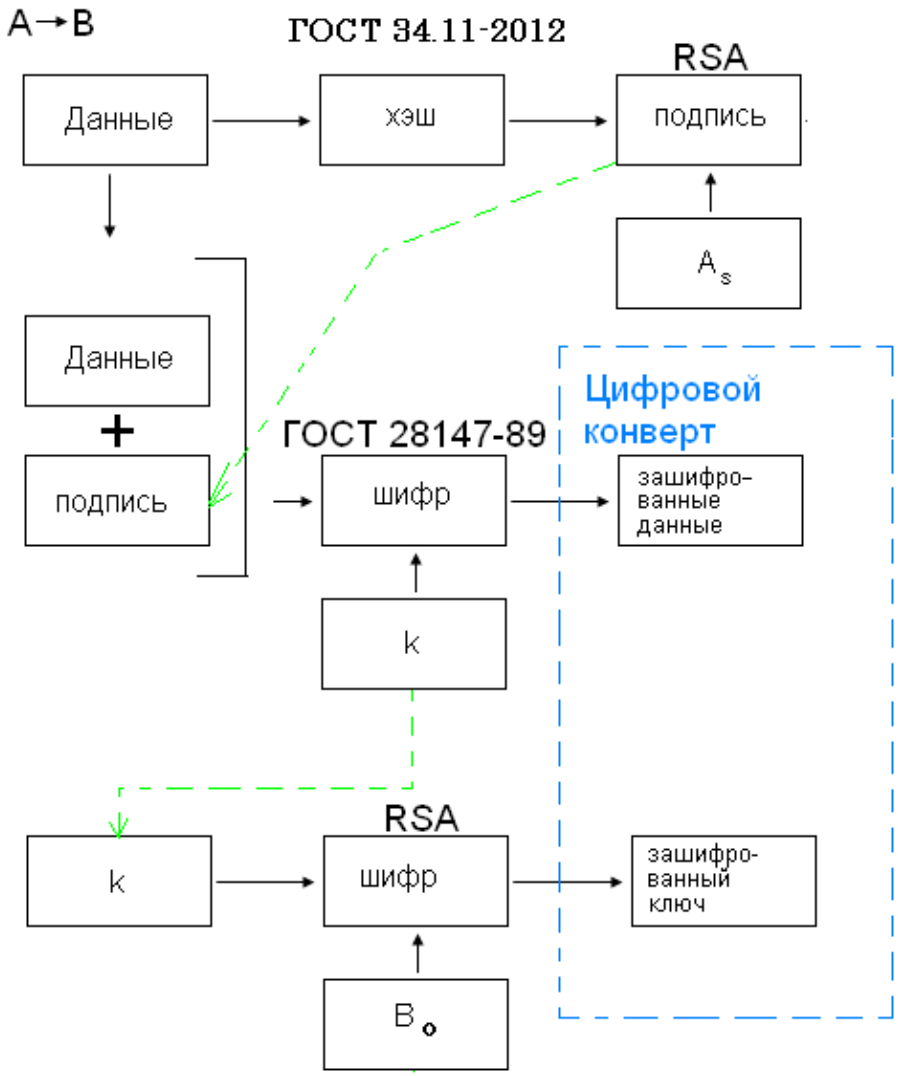

Рисунок 7.1*. Схема отправки зашифрованных данных.*

Варианты заданий:

Вариант 1: - как на рисунках 7.1 и 7.2.

Вариант 2:

- шифрование: DES
- хэш: MD5
- подпись: Эль-Гамаля

Вариант 3:

- шифрование: Blowfish
- хэш: ГОСТ Р 34.11-94
- подпись: RSA

Вариант 4:

- шифрование: AES
- хэш: SHA-2
- подпись: Эль-Гамаля

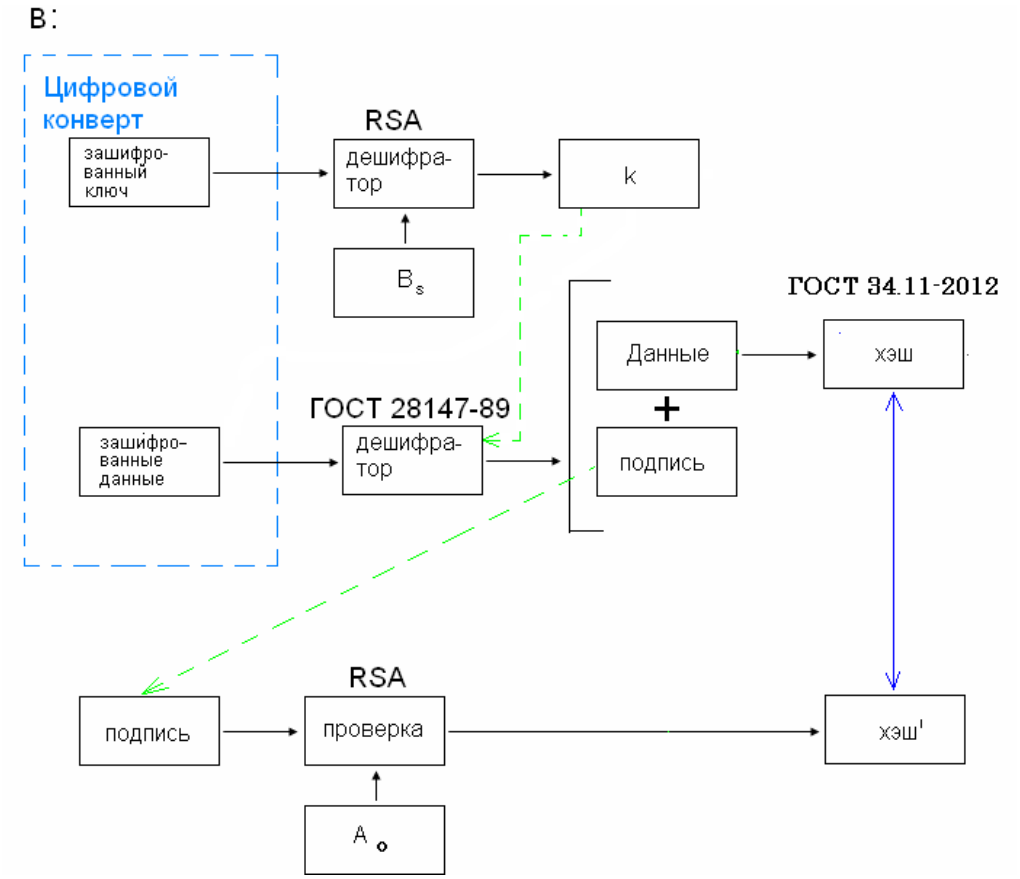

Рисунок 7.2. *Схема приема зашифрованных данных.*

Контрольные вопросы:

1. Принцип работы алгоритма шифрования DES
2. Принцип работы алгоритма шифрования Blowfish
3. Принцип работы алгоритма шифрования AES
4. Зачем ввели в пользование ЭЦП
5. Какими свойствами должна обладать ЭЦП
6. Область применения хэш-функций
7. Требования, предъявляемые к хэш-функциям

Литература

1. В. В. Ященко Введение в криптографию. — М.: МЦНМО-ЧеРо, 2000.
2. Шнайер, Б. Прикладная криптография. Протоколы, алгоритмы, исходные тексты на языке Си / Б. Шнайер – М.: ТРИУМФ, 2002. – 816 с.

Лабораторная работа №8

Защита конфиденциальных данных на ПК и сменных носителях с помощью TrueCrypt.

Цель работы: Изучить принципы управления ключами на основе электронных сертификатов и архитектуру инфраструктуры открытых ключей. Научиться применять криптографические средства защиты конфиденциальных данных на ПК и сменных носителях.

Теоретическая часть [1]:

TrueCrypt это программное обеспечение, предназначенное для создания томов (устройств хранения данных) и работы с ними с использованием шифрования на лету (*on-the-fly encryption*). Шифрование на лету означает, что данные автоматически зашифровываются непосредственно перед записью их на диск и расшифровываются сразу же после их загрузки, т. е. без какого-либо вмешательства пользователя. Никакие данные, хранящиеся в

зашифрованном томе, невозможно прочитать (расшифровать) без правильного указания пароля/ключевых файлов или правильных ключей шифрования. Полностью шифруется вся файловая система (имена файлов и папок, содержимое каждого файла, свободное место, метаданные и др.).

Файлы можно копировать со смонтированного тома *TrueCrypt* и на него точно так же, как и при использовании любого обычного диска (например, с помощью перетаскивания).

При чтении или копировании из зашифрованного тома *TrueCrypt* файлы автоматически на лету расшифровываются (в память/ОЗУ). Аналогично, файлы, записываемые или копируемые в том *TrueCrypt*, автоматически на лету зашифровываются в ОЗУ (непосредственно перед их сохранением на диск). Обратите внимание, это не означает, что перед шифрованием/дешифрованием в ОЗУ должен находиться весь обрабатываемый файл.

Поэтапный процесс создания и использования контейнера *TrueCrypt* подробно рассмотрен в [1].

Задание: В ходе выполнения задания необходимо установить утилиту криптографической
защиты информации *TrueCrypt* (http://www.truecrypt.org/downloads), создать защищенный контейнер для хранения конфиденциальной информации на жестком диске и сменном носителе.

Порядок выполнения:

1. Скачать утилиту *TrueCrypt* с сайта http://www.truecrypt.org/downloads и установить (для установки требуются права администратора).
2. Настроить защищенный файловый контейнер для хранения конфиденциальной информации на жестком диске.
3. Настроить защищенный контейнер на сменном носителе.

В отчете привести порядок настройки защищенного контейнера в TrueCrypt в

форме экранных.

Содержание отчета:

1. Титульный лист
2. Задание
3. Скриншоты, иллюстрирующие порядок настройки защищенных контейнеров в TrueCrypt
4. Выводы по возможностям TrueCrypt

Контрольные вопросы:

1. Что такое TrueCrypt?
2. Что означает шифрование «на лету»?
3. Какие криптографические задачи решает TrueCrypt?
4. Опишите процесс доступа к файлу, хранящемуся в томе TrueCrypt.

Литература

1. Руководство пользователя TrueCrypt, версия 7.1a.

Лабораторная работа №9

Шифрование данных на жестком диске при помощи системы GnuPG.

Цель работы: Ознакомиться с возможностями системы *GnuPG* для защищенного хранения файлов на жестком диске

Теоретическая часть: *PGP* (*Pretty Good Privacy*) – это криптографическая программа, позволяющая проводить операции шифрования и дешифрования файлов, а также электронной подписи. *GnuPG* (*GNU Privacy Guard*, Страж приватности *GNU*) — это свободный некоммерческий аналог *PGP*, как и *PGP*, основанный на *IETF*-стандарте *OpenPGP* [1].

OpenPGP — это стандарт, выросший из программы *PGP*, получившей в Интернете к середине 90-х повсеместное распространение как надёжное

средство шифрования электронной почты. Став стандартом де-факто, *PGP* начал встраиваться во множество приложений и систем. [1]

Порядок выполнения: Рассмотрим работу с *GnuPG* в операционной среде *Ubuntu*. Для работы потребуется *Ubuntu* версии 10.04 или выше. Пакет для *GPG* в этих операционных системах инсталлируется при установке системы. Поэтому можно начинать пользоваться *GnuPG* без предварительной доустановки пакетов *GnuPG*.

Сначала создадим ключи, которые нам потребуются для шифрования и подписи. Для этого в терминале необходимо ввести команду:

```
$ gpg --gen-key
```

GnuPG спросит какой тип ключа вы хотите создать:

(1) RSA and RSA (default)

(2) DSA and Elgamal

(3) DSA (sign only)

(4) RSA (sign only)

Выберем пункт (1). Далее программа попросит указать размер ключа. Рекомендуется указать максимально большую длину ключа: 4096 – для увеличения надежности шифрования.

Далее программа спросит о времени жизни создаваемого ключа. Выберем вариант, стоящий по умолчанию – неограниченное по времени использование ключа (если вы точно уверены, что создаваемый ключ будет использоваться ограниченное время, то можно указать другой вариант).

Далее последует вопрос о вашем имени и *email*. Введите их. После этого программа предложит написать комментарий: писать его не обязательно, но вы можете указать его если считаете, что он может помочь вам в будущем ориентироваться в множестве созданных ключей.

Далее программа попросит ввести парольную фразу (т. е. любой надежный пароль, в котором могут присутствовать пробелы).

После того, как вы введете все запрашиваемые данные, программа начнет генерировать ключи, что может занять несколько минут. После

завершения процесса генерации ключей вы получите сообщение наподобие нижеприведенного:

```
pub   2048R/2E8F71E6 2013-06-24
Key fingerprint = 343D 6A32 EB8C 4995 4C9A  22EC 0206 EE26 2E8F 71E6
uid                  test_name (no comment) <test@mail.ru>
sub   2048R/561FDA97 2013-06-24
```

Теперь, когда вы создали пару открытый и закрытый ключ, вы должны передать свой открытый ключ человеку или группе людей, от которых собираетесь получать зашифрованные на этом ключе сообщения. Для этого вы можете опубликовать свой ключ в Интернете, либо передать его каким-нибудь удобным для вас способом (например, передать на флешке).

Чтобы экспортировать свой открытый ключ в директорию, в которой вы находитесь, в терминале вбейте следующую команду:

```
gpg --armor --output user_name.asc --export user_name ,
```

где user_name – ваше имя, которое вы указывали на этапе генерации ключей. В результате выполнения этой команды в текущей директории появится файл user_name.asc, хранящий ваш открытый ключ. Этот файл вы и должны любым удобным способом передать людям, от которых собираетесь получать шифрованные сообщения.

В тоже время человек, который собирается от вас получать шифрованные сообщения, должен проделать точно такие действия и предоставить вам свой открытый ключ. Предположим, что этот ключ он назвал other_user_name.asc. Теперь вам необходимо импортировать предоставленный вам ключ в базу ключей *GnuPG*. Для этого в терминале вбейте следующую команду:

```
gpg --import other_user_name.asc
```

Просмотреть все имеющиеся у вас ключи вы можете с помощью следующей команды:

```
gpg --list-keys
```

Если вам потребовалось удалить какой-либо ключ, используйте команду:

gpg --delete-secret-and-public-key user_name

После того, как вы импортировали себе ключ other_user_name.asc, вы должны проверить достоверность ключа, который Вы добавили себе. В случае успешной проверки требуется ключ подписать. Проверка достоверности ключа производится с помощью команды *fpr*, введенной после ввода в терминале команды

gpg --edit-key user_name

После ввода этой команды вы увидите надпись ->Command>, после которой нужно ввести команду *fpr*.

Подписать ключ можно с помощью следующей команды:

->Command> sign

Теперь, когда переданный вам открытый ключ подписан и проверен, вы можете отправить ему любой файл, зашифрованный с помощью его открытого ключа (пусть это будет файл test). Сделать это можно следующей командой:

gpg --output test.asc --encrypt --sign --recipient user_name test

где test.asc – имя зашифрованного файла.

Далее вы можете передавать зашифрованный файл test.asc пользователю “user_name”. Пользователь “user_name” в свою очередь с помощью Вашего открытого ключа может зашифровать какой-нибудь файл и послать его вам. Для того, чтобы вам расшифровать файл other_test.asc, запустите в терминале следующую команду:

gpg --output other_test --decrypt other_test.asc

При выполнении команды, программа потребует у вас парольную фразу, которую нужно будет ввести, после чего в текущей директории появится расшифрованный файл other_test, а в окне терминала появится сообщение, содержащее некоторые данные о подписи отправителя, а также информация о том, что подпись верна:

gpg: Signature made Mon 24 Jun 2013 10:13:49 PM MSK using RSA key ID AB735B42

gpg: Good signature from "other_user <other_user@mail.ru>"

Функциями *GnuPG* можно пользоваться не только через терминал. Имеется также графический интерфейс *KGPG* для работы с *GnuPG*. Для его установки в терминале введите команду:

```
sudo apt-get install kgpg
```

После завершения установки *KGPG* в терминале введите команду для запуска: kgpg

KGPG имеет все основные функции программы *GPG*, работа с некоторыми из них через терминал описывалась выше.

Задание: Создайте с помощью программы *GnuPG* ключи, обменяйтесь с товарищем открытыми ключами. Убедившись в правильности полученных открытых ключей. Обменяйтесь с товарищем зашифрованными и подписанными файлами. Дешифруйте их и убедитесь в истинности отправителя.

Содержание отчета:

1. Титульный лист.
2. Задание.
3. Скриншоты, иллюстрирующие порядок выполнения программы.
4. Выводы по возможностям GPG.

Контрольные вопросы:

1. Что такое *PGP*?
2. Что такое *OpenPGP*?
3. Функциональные возможности программы *GnuPG*?
4. Перечислите имеющиеся функции программы *KGPG*..
5. Какие криптографические алгоритмы используются в *GnuPG*?

Литература

1. Проект "open*PGP* в России" [офиц. сайт] *URL: https://www.pgpru.com* (дата обращения: 25.06.2013).