

Практическая криптография в Java. Создаем HTTPS-сервер

Николай Жишкевич

1. Постановка задачи

В прошлой статье серии я начал рассказ о том как сделать сетевые коммуникации безопасными, как сделать так, чтобы можно было устанавливать сетевые соединения и передавать информацию не боясь того, что ее перехватит злоумышленник, не боясь того что информация будет искажена или специально подделана. В прошлый раз нам удалось разработать довольно неплохую иллюстрацию примера аутентификации пользователя на сервере. Что же давайте продолжим. Сегодня мы поставим и попробуем решить задачу создания веб-сервера работающего по протоколу SSL или TLS. Я уже начинал рассказывать об этих протоколах, а также роли и месте которые они занимают в мире web во второй части данной серии. Сегодня мы продолжим и перейдем от теории к практике создав собственный веб сервер с поддержкой HTTPS. На данный момент мы обладаем всеми необходимыми знаниями для успешной реализации данной задачи. Однако, с целью более качественного изложения я планирую решить ее в два этапа. Первый этап будет состоять в создании "просто веб-сервера", который будет работать по протоколу HTTP и будет уметь возвращать клиенту по запросу просто документы HTML, а также запускать на выполнение скрипты, например, PHP или Perl, etc. После того как эта часть задачи будет решена, я добавлю поддержку и возможность работы по протоколу HTTPS.

Я предполагаю что вы обладаете рядом необходимых для сегодняшнего дня знаний. прежде всего нам потребуется умение работать с сокетами, понимание протокола HTTP, знания о том какие существуют MIME типы документов. Если же вы этого не знаете то прежде чем читать дальше советую обратиться к материалам по этой теме, в списке ресурсов опубликованном в конце части 3 данной серии я рекомендовал

воспользоваться введением в сетевые технологии и, разумеется, что всякая дополнительная литература будет приветствоваться.

Давайте на некоторое время отложим в сторону всякие `Socket`, `ServerSocket`, `URLConnection` и другие классы и подумаем над тем как мы будем управлять работой нашего просто веб-сервера. Очевидно, что жесткое указание в теле кода программы параметров и путей будет и не красиво и не правильно. В качестве формата для хранения настроек можно использовать `Properties`-файлы, параметры командной строки, однако для дальнейшего развития примера этого, пожалуй, будет не хватать. Таким образом, я склоняюсь к использованию в качестве файла настройки документа `xml`. Кроме того есть еще одна причина по которой я сделал данный выбор, надеюсь, вы помните, что документы `xml` являются самоописывающимися, так что мне не придется тратить много времени на описание формата файла конфигурации. Все таки не зря говорят, что программисты это комбинация лени и логики.

Все настройки приложения хранятся в виде файлов `XML`. Для анализа которых я разработал отдельный класс, который сканирует дерево `XML` в поиске нужных компонент. На вход методам данного вспомогательного класса подается строка похожая на выражения `xpath`. Вот пример выражений пути, которые будет уметь обрабатывать мой движок `/organization/sections/section/director, libraries/library[4]/books/book[1245]/author`. Исходный код движка я приведу позже, однако сразу хочу заметить, что он настолько прост, что вы легко сможете разобраться в принципах его работы, в качестве ядра я решил выбрать `herces` от `apache`.

А вот пример файла конфигурации для веб-сервера. На текущий момент к его способностям относится возможность отвечать на запросы веб-клиентов по методу `"GET"` и возвращать документы `HTML`, картинки ну и, пожалуй, еще был реализован простейший механизм работы с серверными расширениями. Далее я приведу код файла конфигурации для сервера. Обратите внимание прежде всего на указание номера порта, который будет слушать сервер, путь к каталогу документов, и как в настоящем серьезном продукте я добавил имя сервера и информацию об администраторе, чтобы в случае возникновения ошибок выводить эти сведения клиенту, чтобы знал, кому писать письма с благодарностями. Также с небольшими планами на расширение я ввел поддержку типов `MIME`, дело в том, что мне в моей

Практическая криптография в Java. Создаем HTTPS-сервер

работе, которая, в конечном счете, и привела к появлению данного раздела данной статьи данной серии статей, была необходимость создания распределенной системы управления справочной системой и иными ресурсами приложений. Ну да ладно, не в том, дело. Итак, вот пример файла конфигурации.

```
<?XML version="1.0" encoding="UTF-8" ?>
- <!--
  edited with XMLSPY v5 rel. 4 U (http://www.xmlspy.com) by Nikolai (Comp)
  -->
- <httpd>
  <Port>1088</Port>
  <ServerName>SimpleServer</ServerName>
  <ServerAdmin>Master (x13@tut.by)</ServerAdmin>
  <DocumentRoot>e:\mdocs\kolya\web_root</DocumentRoot>
  <CGIRoot>e:\mdocs\kolya\web_root\cgi-bin</CGIRoot>
- <Mappings>
- <Mapping>
  <Extension>php</Extension>
  <Application>D:\Program_Files_2\php4\php-4.3.0-Win32\php.exe</Application>
</Mapping>
- <Mapping>
  <Extension>pl</Extension>
  <Application>D:\Program_Files_2\Perl\bin\Perl.exe</Application>
</Mapping>
- <!--
  Хотя в данной версии возможность работы со скриптами
  CGI не реализована в полной мере, но в перспективе
  я обязательно реализую данную возможность,
  пока не очень грамотно идет обработка заголовков
  возвращаемых серверным расширением, ну и прочие мелочи ...
  -->
  </Mappings>
- <Types>
- <Type>
  <Extension>HTML</Extension>
  <Mime-type>text/HTML</Mime-type>
  <Text />
</Type>
- <Type>
  <Extension>htm</Extension>
```

```
<Mime-type>text/HTML</Mime-type>
<Text />
</Type>
- <Type>
  <Extension>jpg</Extension>
  <Mime-type>image/jpeg</Mime-type>
  </Type>
</Types>
</HTTpd>
```

2. Код реализации. HTTP сервер

После того как мы определились по поводу функциональных возможностей данного сервера, давайте подумаем над тем, какие знания нам потребуются для его успешной реализации.

Прежде всего, необходимо будет создать класс для анализа документов XML – т.е. для анализа нашего файла конфигурации. Что же как я и обещал выше привожу его. Движок конечно доморощенный, но работает вполне сносно, а что же от него еще и желать. Дело в том, что конечно более правильным было бы его создать на основе XPath API или regex (работа с регулярными выражениями, наконец начиная с JSDK1.4 появилась, так что нет необходимости выкачивать отдельно библиотеки вроде `apache.org`, спасибо Sun-у, постаралась). Однако, подобный ход потребовал бы еще одного отвлечения от главной темы что автоматически повысило требования к наличным знаниям читателя. Так что как мне не хотелось заменить эту первоначальную версию движка на что-нибудь более "продвинутое", пожалуй, останусь на своем.

```
package arti.security;

import org.w3c.dom.*;
import java.util.*;

public class XTools {

    public static Element getElementByXPath(String xp, Element from) {
        while (xp.startsWith("/")) {
            // избавляемся от лидирующих символов "/"
            xp = xp.substring(1);
        }
    }
}
```

Практическая криптография в Java. Создаем HTTPS-сервер

```
}
StringTokenizer tok = new StringTokenizer(xp, "@");
if (!tok.hasMoreTokens()) {
    return null;
}
String ce = tok.nextToken();
String cce = ce;

if (ce.indexOf("[") != -1) {
    // выделяем только имя элемента
    ce = ce.substring(0, ce.indexOf("["));
}
if (!ce.equals(from.getNodeName())) {
    return null;
}

if (!tok.hasMoreTokens()) {
    return from;
}

String ce2 = tok.nextToken();

if (ce2.indexOf("[") == -1) {
    NodeList child = from.getChildNodes();
    int cnt = child.getLength();
    for (int i = 0; i < cnt; i++) {
        Element elt = null;
        try {
            elt = (Element) child.item(i);
        }
        catch (Exception ex) {continue;}
        Element candi = getElementByXPath(
            xp.substring(cce.length() + 1), elt);
        if (candi != null) {
            return candi;
        }
    }
    return null;
}
```

```
// в том случае если есть нумерация элементов с одинаковыми именами

String idx = ce2.substring(ce2.indexOf("[") + 1);
// теперь в idx находится
// только номер элемента, ничего более
idx = idx.substring(0, idx.length() - 1);
int iidx = Integer.parseInt(idx);
NodeList child = from.getChildNodes();
int cnt = child.getLength();
// выделяем только имя элемента
String ce3 = ce2.substring(0, ce2.indexOf("["));
int c_sel = 0;
for (int i = 0; i < cnt; i++) {

    Element elt = null;
    try {
        elt = (Element) child.item(i);
    }
    catch (Exception ex1) {
        continue;
    }
    if (elt.getNodeName().equals(ce3)) {
        if (c_sel++ == iidx) {
            return getElementByXPath(
                xp.substring(ce.length() + 1), elt);
        }
    }
}
return null;
}

public static String getXPathText (String xp, Element from){
    Element el = getElementByXPath(xp , from);
    if (el == null) return null;
    // элемент пустой и не имеет вложенного содержимого
    if (el.getFirstChild() == null) return "";
    return el.getFirstChild().getNodeValue();
}

public static Integer getXPathInt (String xp, Element from){
```

Практическая криптография в Java. Создаем HTTPS-сервер

```
Element el = getElementByXPath(xpath , from);
if (el == null) return null;
return new Integer (
    el.getFirstChild().getNodeValue().trim());
}

public static Double getXPathDouble (String xpath, Element from){
    Element el = getElementByXPath(xpath , from);
    if (el == null) return null;
    return new Double (
        el.getFirstChild().getNodeValue().trim());
}
}
```

После того как я привел код файла настроек и пример кода движка для разбора данного документа, неплохо было бы привести и код "просто сервера". Что же, привожу. Однако обратите внимание на фрагмент кода для функции получения объекта серверного сокета, более подробный комментарий и пояснение его роли будет раскрыто далее.

```
package arti.security;

import java.io.*;
import java.net.*;
import org.apache.xml.utils.*;

import org.xml.sax.helpers.*;
import org.w3c.dom.*;

/**
 * Данный класс определяет функциональность
 * для веб-сервера в нем реализована возможность
 * настройки через файл конфигурации в формате XML
 * на настоящий момент релизована возможность
 * обработки серверных расширений,
 * автор протестировал их на основе php
 * большая часть функциональности реализована,
 * не доведена до конца поддержка cookie , post-запросов.
 * Однако следует помнить, что в данном случае
 * я ставил перед собой совсем иную задачу
```

Практическая криптография в Java. Создаем HTTPS-сервер

```
*/
public class AbstractServer {
// номер локального порта, к которому присоединился наш веб-сервер
    protected int localPort;
// имя файла конфигурации
    protected String fNameConfig;
// каталог в котором находятся документы
    protected String localWebRoot;
// каталог в котором находятся исполняемые приложения
    protected String localCGIRoot;

// имя сервера
    protected String serverName;
// информация об администраторе,
//чтобы была возможность писать гневные письма :)
    protected String AdminInfo;
// ссылка на дерево DOM построенное на основе файла конфигурации
    protected org.w3c.dom.Document configDoc;

    public AbstractServer(String fName) throws IOException {
        fNameConfig = fName;
    }

    ServerSocket sock;// сокет сервера
    public boolean prepareForRun() {
        // данная функция служит для того
        // чтобы провести начальную инициализацию
        // сервера, прочитать параметры его работы из файла конфигурации
        // стоит сказать что приведенный фрагмент кода не является идеальным
        // я вижу возможное направление дальнейшей оптимизации в
        // перестройке информации прочтенной из файла
        // конфигурации быть может не в столь удобную форму,
        // но более быстро работающую
        try {
            org.apache.xerces.parsers.DOMParser dp =
                new org.apache.xerces.parsers.DOMParser();
            dp.parse(new org.XML.sax.InputSource(
                new InputStreamReader(new FileInputStream(fNameConfig),
                    "utf-8")));
            org.w3c.dom.Document doc = dp.getDocument();

```

Практическая криптография в Java. Создаем HTTPS-сервер

```
configDoc = doc;
Integer Ip = XTools.getXPathInt(
    "httpd/Port", doc.getDocumentElement());
if (Ip == null) {
    System.out.println(
        "Критическая ошибка файла конфигурации,
        отсутствует номер порта для сервера");
    return false;
}
localPort = Ip.intValue();
localWebRoot = XTools.getXPathText("HTTPd/DocumentRoot",
    doc.getDocumentElement());
localCGIRoot = XTools.getXPathText("HTTPd/CGIRoot",
    doc.getDocumentElement());

serverName = XTools.getXPathText("HTTPd/ServerName",
    doc.getDocumentElement());
AdminInfo = XTools.getXPathText("HTTPd/ServerAdmin",
    doc.getDocumentElement());

System.out.println("Info: Server: " +
    serverName + " at " + localPort +
    " ; alias for root = " + localWebRoot +
    "; alias for cgi = " + localCGIRoot +
    " ;admininfo = " +
    AdminInfo);

if (
    (serverName == null) ||
    (localCGIRoot == null) ||
    (localWebRoot == null) ||
    (AdminInfo == null)
    ) {
    System.out.println("Неполная информация : запуск сервера невозможен");
    return false;
}
}
catch (Exception ex) {
    ex.printStackTrace();
    return false;
}
```

Практическая криптография в Java. Создаем HTTPS-сервер

```
// грубо но пожалуй для иллюстрации сойдет вполне
// разумеется, что более правильным было бы
// обработка каждого исключения в отдельности или их
// перевыборос, однако ... думаю это тоже неплохое решение
}
return true;

}

public void synchoRun() {
    System.out.println("Start Listen at " + localPort);
    try {
        sock = createCoreSocket();
    }
    catch (IOException ex1) {
        sock = null;
    }
    // обратите большое внимание на данный фрагмент кода
    // здесь я вызываю обобщенную
    // функцию создания сокета на стороне сервера
    // если мы создадим класс производный от данного
    // и изменим поведение данной функции, то возможно
    // произвольное расширение возможностей приложения
    // практически не затрагивая весь объем написанного кода

    if (sock == null){
        System.out.println("Ошибка: не могу создать сокет для сервера");
        return;
    }
    while (true) {

        try {
// получаем сокет описывающий запрос со стороны клиента
            final Socket cs = sock.accept();
            System.out.println("Start New Session for client [" + cs + "]");
            new Thread() {
                public void run() {
                    OutputStream cliout = null;
                    try {
                        DataInputStream din = new DataInputStream(cs.getInputStream());
```

Практическая криптография в Java. Создаем HTTPS-сервер

```
String s, uri = null;
// пример запроса HTTP будет выглядеть примерно так
// "GET /me/doc.HTML HTTP/1.1"
// Заголовки завершаются пустой строкой
while ( (s = din.readLine()) != null && s.length() != 0) {
    System.out.println("log: клиент говорит:" + s);
    if (s.toUpperCase().startsWith("GET")) {
        uri = s;
    }
}
if (uri == null) {
    System.out.println("Неверный запрос документа");
    cs.close();
    return;
}

String doc = uri.substring(uri.indexOf(" ") + 1,
                           (uri.lastIndexOf(" ") -
                            uri.indexOf(" ") + 3));
System.out.println("Try Process \"" + doc + "\"");
String fullName = localWebRoot + doc;

printFile(fullName, cliout = cs.getOutputStream());
cs.close();
}
catch (IOException ex) {
    ex.printStackTrace();
    PrintStream pout = new PrintStream (cliout);
    pout.println("Внутренняя ошибка сервера <hr/>");
    ex.printStackTrace(pout);
}

}
}
.start();
}
catch (IOException ex) {
    ex.printStackTrace();
}
}
```

Практическая криптография в Java. Создаем HTTPS-сервер

```
    }  
}  
  
public void printFile(String fullName,  
    OutputStream out) throws IOException {  
    File f = new File(fullName);  
    System.out.println("Local: " + f.getAbsolutePath());  
    if (!f.exists() || f.isDirectory()) {  
        PrintStream ps = new PrintStream(out);  
        ps.println("Ошибка, запрошенный вами ресурс не найден на сервере <br/>");  
        ps.println("<hr/>");  
        ps.println("AbstractServer at " + localPort + "; admin " + AdminInfo);  
        return;  
    }  
    String fsn = f.getAbsolutePath();  
    String ext = fsn.substring(fsn.lastIndexOf(".") + 1);  
    String mime = getMimeType(ext);  
    if (isTextDocument(mime)) {  
        // в общем случае механизм действий  
        // по обработке документа типа текст  
        // и произвольного другого документа  
        // может иметь отличия  
        System.out.println(  
            "Text: Документ который был запрошен является текстовым");  
  
        BufferedReader br = new BufferedReader(new InputStreamReader(  
            processMatches(fullName)));  
        String s;  
        PrintStream ps = new PrintStream(out);  
        //ps.println("Content-Type: "+ mime+"\n");  
  
        while ( (s = br.readLine()) != null) {  
            ps.println(s);  
        }  
        ps.flush();  
        return;  
    }  
  
    System.out.println(  
        "Non-Text: Документ который был запрошен не является текстовым");  
}
```

Практическая криптография в Java. Создаем HTTPS-сервер

```
FileInputStream fin = new FileInputStream(fullName);
byte[] buf = new byte[10000];
int rsize;
while ( (rsize = fin.read(buf)) > 0) {
    out.write(buf, 0, rsize);
}

}

protected boolean isTextDocument(String mime) {
    if (mime == null) {
// по умолчанию будем считать,
// что возвращается текстовой документ
        return true;
    }
    Element root = configDoc.getDocumentElement();
    for (int i = 0; ; i++) {
        //System.out.println("Определение типа документа
            на основании его MIME-типа");
        String mime_candi =
            XTools.getXPathText("httpd/Types/Type[" + i +
                "]/Mime-type", root);

        if (mime_candi == null) {
// вообще-то более правильным было бы выбросить отсюда исключение, но
            return true;
        }
        // пожалуй это потом доработаю
        if (mime.equalsIgnoreCase(mime_candi)) {
            return
                (XTools.getXPathText("HTTPd/Types/Type[" + i +
                    "]/Text", root) != null); // если существует элемент
        }
    }
}

protected String getMimeType(String ext) {
    Element root = configDoc.getDocumentElement();
    for (int i = 0; ; i++) {
```

Практическая криптография в Java. Создаем HTTPS-сервер

```
//System.out.println("Определение MIME-типа документа
    на основании его расширения");
String ext_candi = XTools.getXPathText("HTTPd/Types/Type[" + i +
    "]/Extension", root);

if (ext_candi == null) {
    return null;
}
if (ext.equalsIgnoreCase(ext_candi)) {
    return XTools.getXPathText("HTTPd/Types/Type[" + i + "]/Mime-type",
        root);
}
}
}

// Данная функция получает на вход имя файла, проверяет
// является ли данный файл серверным расширением,
// если это так, то запускает его на выполнение
// и результат работы возвращает в выходном потоке
// если же это не так, то просто возвращается документ

protected InputStream processMatches(String fname) throws IOException {
    String fext = fname.substring(fname.lastIndexOf(".") + 1);
    Element root = configDoc.getDocumentElement();
    for (int i = 0; ; i++) {
        String ext_candi = XTools.getXPathText("HTTPd/Mappings/Mapping[" + i +
            "]/Extension", root);

        if (ext_candi == null) {
            break;
        }

        if (ext_candi.equalsIgnoreCase(fext)) {
            String aplName = XTools.getXPathText("HTTPd/Mappings/Mapping[" + i +
                "]/Application", root);

            System.out.println("Приложение для запуска серверного расширения = \"\" +
                aplName + "\"");
            if (aplName.length() < 2) {
                aplName = "";
            }
            else {
                aplName += " ";
            }
        }
    }
}
```

Практическая криптография в Java. Создаем HTTPS-сервер

```
    }
    aplName+= fname;
    System.out.println("Полная строка запуска
        серверного расширения=\"\" + aplName+ \"\");

    Process pro = Runtime.getRuntime().exec(aplName);
    try {
        ByteArrayOutputStream b = new ByteArrayOutputStream ();
        byte [] mb = new byte [10000];
        int rsize;
        InputStream inP = pro.getInputStream();
        while ((rsize = inP.read(mb))> 0)
            b.write (mb , 0 , rsize);

        pro.destroy();
        System.out.println("Серверное приложение завершило работу");
        return new ByteArrayInputStream (b.toByteArray());
    }
    catch (Exception ex) {
        ByteArrayOutputStream b;
        PrintStream ps = new PrintStream(b = new ByteArrayOutputStream());
        ex.printStackTrace(ps);
        ps.flush();
        return new ByteArrayInputStream(b.toByteArray());
    }

}

}
return new FileInputStream(fname);
}

protected Socket modifySocket (Socket s) throws IOException{
// на основе данной функции и следующей
// будет создано дальнейшее расширение поддерживающее работу с SSL
    return s;
}

protected ServerSocket createCoreSocket () throws IOException{
```

```
return new ServerSocket (localPort);  
}  
}
```

3. Анализ исходного кода.

Давайте рассмотрим исходный код примера. Я разработал класс `AbstractServer` который и реализует функциональность "просто" сервера. При создании экземпляра данного класса с помощью конструктора в качестве параметра ему необходимо будет передать строку содержащую имя файла настроек. Непосредственный разбор данного файла и чтение параметров (имя сервера, номер порта который будет обслуживать наш сервер, физическую директорию в которой будут располагаться документы и другие) выполняется в методе `prepareForRun`. После выполнения данного метода если конечно он завершается успешно (в качестве признака используется возврат логического значения) сервер готов к работе. Непосредственный его запуск выполняется с помощью метода `synchroRun`. В данном методе создается серверный сокет и мы внутри цикла начинаем ждать соединения клиентов. После соединения идет чтение из входного потока от клиента его запроса. Пока как я и упоминал ранее существует поддержка только метода запроса "GET". Браузер клиента передает набор строк описывающих что ему нужно, в каком формате можно вернуть результат, например можно вернуть информацию в сжатом виде, также часто передается информация о локали, версии и названии браузера клиента и другие параметры. Запрос завершается пустой строкой. Поэтому созданный веб-сервер читает запрос выделяет из него имя документа, затем определяется тип документа который был запрошен (при выполнении данной операции используются параметры заданные в файле конфигурации). Ниже я привожу пример заголовков запроса:

```
Start New Session for client [Socket[addr=/127.0.0.1,port=4860,localport=1088]]  
  
log: клиент говорит:GET /main.htm HTTP/1.1  
  
log: клиент говорит:User-Agent: Mozilla/4.0  
      (compatible; MSIE 6.0; MSIE 5.5; Windows NT 5.1) Opera 7.02 [en]  
  
log: клиент говорит:Host: comp:1088  
  
log: клиент говорит:Accept: text/html, image/png,
```

```
image/jpeg, image/gif, image/x-xbitmap, */*;q=0.1

log: клиент говорит:Accept-Language: en

log: клиент говорит:Accept-Charset:
      windows-1252, utf-8, utf-16, iso-8859-1;q=0.6, */*;q=0.1

log: клиент говорит:Accept-Encoding: deflate, gzip, x-gzip, identity, */*;q=0

log: клиент говорит:Connection: Keep-Alive

Try Process "/main.htm"

Local: e:\mdocs\kolya\web_root\main.htm
```

Основная работа сосредоточена внутри метода `printFile`. Данный метод проверяет существует ли файл, если нет то выводилась специальная страница с сообщением об ошибке. Если же документ существует, то он читается и возвращается клиенту.

Я уже упоминал, но пожалуй повторюсь, обратите основное внимание на метод `createCoreSocket` данный метод служит для создания серверного сокета. Я вынес данный кусок кода из функции `synchroRun` с планами на то что в будущем данный метод будет перекрыт и именно внутри его будет помещен весь набор действий по обработке запросов по протоколу HTTPS. Ведь согласитесь, что с точки зрения приложения работающего на прикладном уровне модели OSI происходящее на более низких уровнях не существенно. И методу `printFile` в общем случае будет все равно как именно будет выполняться запрос по открытому протоколу HTTP или по закрытому HTTPS. Ведь в любом случае запрос придет в виде "GET имя документа" и в любом случае надо будет просто вернуть прочитанный с диска файл. А то что обмен информацией будет идти через еще один дополнительный слой который будет выполнять прозрачное (самое главное слово) шифрование и проверку целостности информации, то это уже не его забота.

4. Краткие технические сведения. Необходимые классы для создания защищенного сервера

Для дальнейшего развития примера и модификации функции `createCoreSocket`

нам потребуется активно использовать следующие классы и интерфейсы Java Crypto API.

`KeyStore` — данный класс представляет собой некоторое абстрактное хранилище набора ключей (имеется в виду тех самых секретных ключей, раскрывать которые никому нельзя) и сертификатов, устанавливающих соответствия между открытыми ключами и конкретными организациями или частными лицами владеющими данными парами ключей. Причем считается что если сертификат находится у вас в хранилище `KeyStore`, то вы ему доверяете. Действительно, попробуйте добавить сертификат из внешнего файла в файл хранилище, у вас обязательно спросят, "А вы ему точно доверяете". Каждая информационная единица (ключ, или сертификат) хранятся под своим именем (`alias`), и всякая операция (например, запрос сертификата или его программное удаление всегда должно сопровождаться указанием псевдонима), стоит также сказать, что псевдонимы не имеют никакого практического значения вне хранилища, они никоим образом не связаны ни с организацией являющейся владельцем ключа или сертификата, ни с той организацией, которая его выдала.

Создание экземпляра `KeyStore` выполняется не через `new`, а согласно уже привычной нам схеме с обращением к списку провайдеров криптографических услуг зарегистрированных в нашей системе с помощью вызова статического метода `getInstance`. После создания экземпляра хранилища следует выполнить его загрузку из конкретного физического местоположения, в простейшем варианте из файла на жестком диске, при выполнении данной операции можно указать пароль для доступа к хранилищу. Надо сказать также, что хотя пароль является не обязательным и его можно не указывать, но это приводит к потере функциональности, так вы уже не можете выполнять операции именно над ключами, и кроме того не может быть проверена целостность самого хранилища. В качестве несложной демонстрации работы с хранилищами ключа я приведу ниже пример программки, которая формирует список сертификатов и позволяет выполнять простейшие действия: "Добавление", "Удаление", "Просмотр".

`TrustManager` — базовый интерфейс, на основе которого строятся иные менеджеры доверия, а действия которые им выполняются очевидны — это принятие решения о доверии ли не доверии к сертификату который нам предоставляют при соединении с удаленной системой по протоколу SSL или TLS. Производный интерфейс

Практическая криптография в Java. Создаем HTTPS-сервер

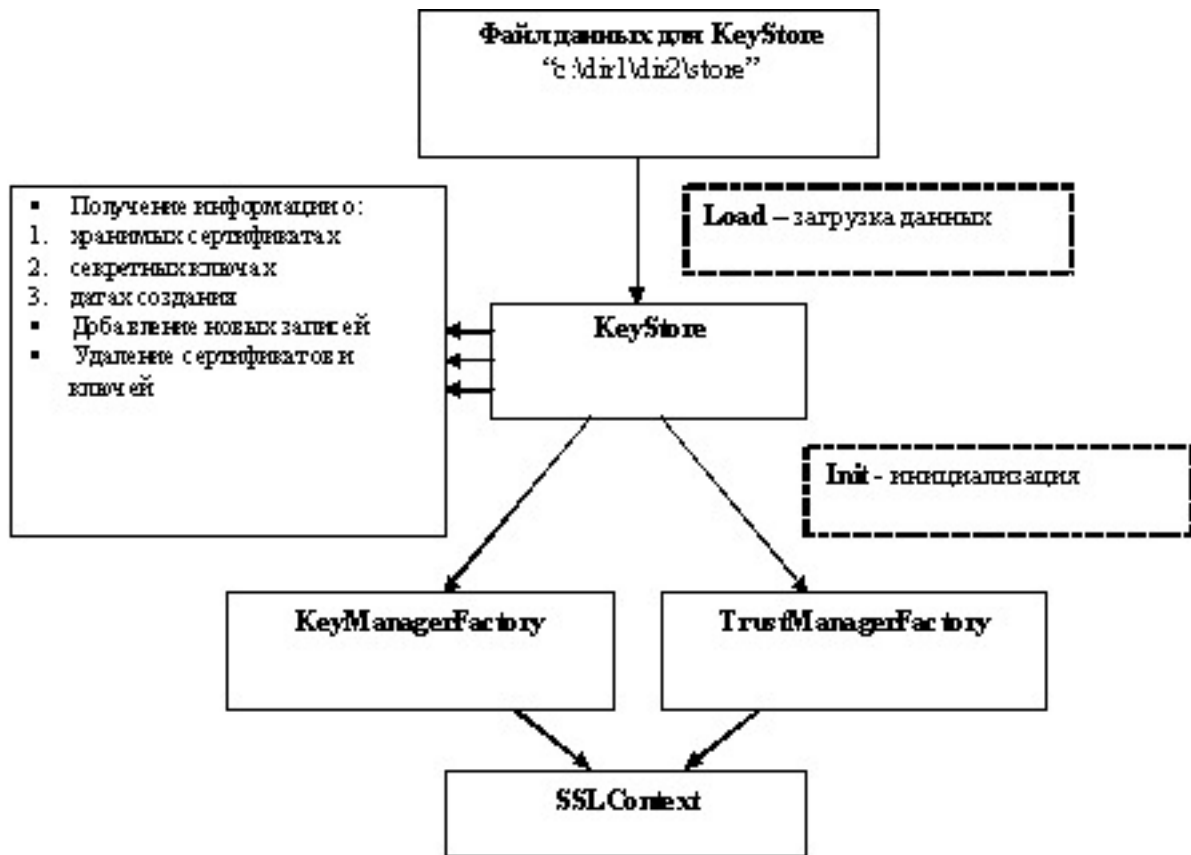
`X509TrustManager` — работа с сертификатами X509. Никогда при создании собственного кода по установлению соединений с защищенными приложениями не забывайте создать набор подобных менеджеров, иначе будет постоянно получать сообщения о невозможности установить соединение с ненадежными источником.

`KeyManager` — интерфейс, классы созданные на основе которого должны обладать способностью к самопредставлению при установлении защищенных соединений. Если в приложении не будет создан набор менеджеров ключей то невозможно создать защищенный сервер, он просто не сможет предоставить клиенту сертификат и не сможет выполнять шифрование данных и аутентификацию.

`TrustManagerFactory` — название данного класса говорит само за себя — он служит для создания набора менеджеров доверия на основании некоторой информации из хранилища `KeyStore`.

`KeyManagerFactory` — фабрика для создания наборов менеджеров ключей.

Таким образом, обобщенная схема использование хранилища и менеджера доверия и менеджера ключей для последующего создания сессий SSL соединений выглядит примерно так:



5. Закрепляем знания о Java Crypto API. Программа просмотра сведений о сертификатах

Как я и обещал ранее мы попробуем закрепить знания по использованию классов KeyStore, Certificate и других небольшим примером. Приведенное далее приложение должно получать информацию о хранилище сертификатов. В общих словах схему его работы можно представить в виде ряда шагов:

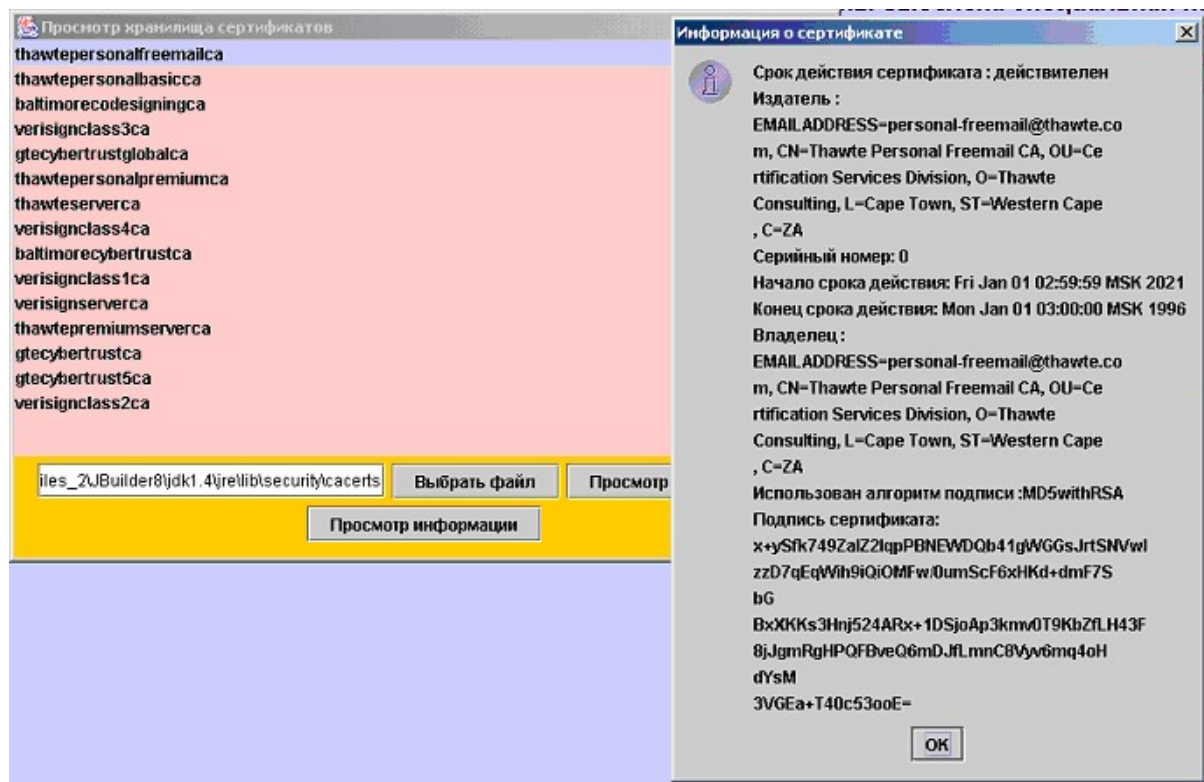
1. Получить имя файла хранилища KeyStore. Имя либо вводится в текстовое поле или выбирается с помощью диалога открытия файла с использованием класса JFileChooser.
2. Создать экземпляр KeyStore, загрузить в него содержимое из файла полученного на предыдущем шаге. Затем с помощью метода aliases получем перечисление,

Практическая криптография в Java. Создаем HTTPS-сервер

содержащее имена всех элементов (сертификатов) хранящихся в данном KeyStore. Все полученные имена заносятся в вектор который используется как модель данных для элемента управления список.

3. После того как пользователь выберет какое-нибудь из имен в списке и нажмет кнопку "Просмотреть сертификат" программа определяет имя сертификата и запрашивается класс сертификата содержащий его описание с помощью вызова метода `KeyStore.getCertificate`. После этого я получаю информацию о содержимом сертификата, кто кому и на какой срок выдал и отображаю информацию в виде `JOptionPane.showMessageDialog`.

Вот исходный код примера. Я постарался добавить необходимое количество комментариев. Прежде чем приступим к непосредственной реализации кода сервера с поддержкой SSL попробуйте разобраться в приведенном ниже примере. Посмотрите на методику работы с классом `KeyStore` и получением статистической информации о сертификатах.



А вот и исходный код примера.

```
package arti.security;

import java.io.*;
import java.security.*;
import java.security.cert.*;
import java.security.cert.Certificate;
import java.util.*;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class SimpleCertificateViewer extends JFrame {
    KeyStore ks; // хранилище ключей и сертификатов
    BorderLayout borderLayout1 = new BorderLayout();
    // инструментарий для создания GUI
    JPanel paneNavi = new JPanel();
    JButton btnBrowseStore = new JButton();
    JTextField txtFileName = new JTextField();
    JButton btnShowAliases = new JButton();
    JList lstAliases = new JList();
    // список в котором будут размещаться
    // псевдонимы для прочитанных сертификатов
    JButton btnShowInfo = new JButton();

    public SimpleCertificateViewer() {
        try {
            jbInit();
        }
        catch (Exception e) {
            e.printStackTrace();
        }
    }

    public static void main(String[] args) {
        SimpleCertificateViewer simpleCertificateViewer =
            new SimpleCertificateViewer();
        simpleCertificateViewer.setSize(600 , 400);
    }
}
```

Практическая криптография в Java. Создаем HTTPS-сервер

```
simpleCertificateViewer.show();
}
private void jbInit() throws Exception {
    this.getContentPane().setBackground(
        UIManager.getColor("ToolBar.dockingForeground")
    );
    setTitle("Просмотр хранилища сертификатов");
    this.getContentPane().setLayout(borderLayout1);
    btnBrowseStore.setText("Выбрать файл");
    btnBrowseStore.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(ActionEvent e) {
            btnBrowseStore_actionPerformed(e);
        }
    });
    txtFileName.setMinimumSize(new Dimension(6, 100));
    txtFileName.setPreferredSize(new Dimension(250, 25));
    txtFileName.setText("Введите имя файла");
    btnShowAliases.setText("Просмотр сертификатов");
    btnShowAliases.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(ActionEvent e) {
            btnShowAliases_actionPerformed(e);
        }
    });
    paneNavi.setBackground(Color.orange);
    lstAliases.setBackground(new Color(255, 201, 201));
    lstAliases.setMaximumSize(new Dimension(600, 300));
    lstAliases.setMinimumSize(new Dimension(600, 300));
    lstAliases.setPreferredSize(new Dimension(600, 300));
    lstAliases.setRequestFocusEnabled(true);
    lstAliases.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
    btnShowInfo.setText("Просмотр информации");
    btnShowInfo.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(ActionEvent e) {
            btnShowInfo_actionPerformed(e);
        }
    });
    this.getContentPane().add(paneNavi, BorderLayout.CENTER);
    paneNavi.add(txtFileName, null);
    paneNavi.add(btnBrowseStore, null);
    paneNavi.add(btnShowAliases, null);
```

```
paneNavi.add(btnShowInfo, null);
this.getContentPane().add(lstAliases, BorderLayout.NORTH);

setDefaultCloseOperation(EXIT_ON_CLOSE);
}

void btnBrowseStore_actionPerformed(ActionEvent e) {
    //отображаем диалог открытия файла и имя выбранного файла
    //помещаем в текстовое поле
    JFileChooser chooser = new JFileChooser();
    int rv = chooser.showOpenDialog(this);
    if(rv == JFileChooser.APPROVE_OPTION) {
        txtFileName.setText(chooser.getSelectedFile().getAbsolutePath());
    }
}

void btnShowAliases_actionPerformed(ActionEvent e) {
    try {
        // загружаем хранилище сертификатов, имя файла берем
        // из текстового поля "txtFileName"
        ks = KeyStore.getInstance(KeyStore.getDefaultType());
        ks.load(new FileInputStream(txtFileName.getText()), null);
        Enumeration E = ks.aliases();
        // получаем перечисление имен псевдонимов "alias"
        // и размещаем их внутри вектора
        Vector v = new Vector();
        while (E.hasMoreElements()) {
            v.add( (String)E.nextElement() );
        }
        lstAliases.setListData(v); // помещаем вектор строк
        // (псевдонимов сертификатов) в список
        JOptionPane.showMessageDialog(this ,
            "Успешно прочитан список сертификатов из "+ v.size() + " элементов");
        invalidate();
    }
    catch (Exception ex) {
        JOptionPane.showMessageDialog(this ,
            "Ошибка на стадии чтения хранилища сертификатов:\n"+ ex);
    }
}
```

Практическая криптография в Java. Создаем HTTPS-сервер

```
void btnShowInfo_actionPerformed(ActionEvent e) {
    Certificate cert = null;
    try {
        // получаем ссылку на выделенную строку с именем сертификата
        cert = ks.getCertificate( (String) lstAliases.getSelectedValue());
    }
    catch (KeyStoreException ex1) {
        JOptionPane.showMessageDialog(this ,
            "Ошибка получения их хранилища сертификата под псевдонимом <<" +
            lstAliases.getSelectedValue()+">>");
        return;
    }
    X509Certificate xcert = (X509Certificate) cert;
    // далее идет фрагмент кода в котором я получаю статистическую
    // информацию о полях данного сертификата
    // (кто, кому выдал, срок действия и т.д.)
    String inf = "Срок действия сертификата : ";
    try{
        xcert.checkValidity();
        inf += "действителен";
    }
    catch (Exception ex){
        inf+="не действителен";
    }
    inf+= "\n";

    inf+="Издатель : " + wrapper(xcert.getIssuerDN().getName()) + "\n";
    inf+="Серийный номер: "+xcert.getSerialNumber() + "\n";
    inf+="Начало срока действия: "+xcert.getNotAfter() + "\n";
    inf+="Конец срока действия: "+xcert.getNotBefore() + "\n";

    inf+="Владелец : " + wrapper(xcert.getSubjectDN().getName()) + "\n";

    inf+="Использован алгоритм подписи : " + xcert.getSigAlgName() + "\n";
    inf+="Подпись сертификата: "+
        wrapper(new sun.misc.BASE64Encoder().encode( xcert.getSignature()));

    JOptionPane.showMessageDialog(this , inf ,
        "Информация о сертификате" , JOptionPane.INFORMATION_MESSAGE);
}
```

```
}  
  
// вспомогательная функция которая выполняет разбиение строки на  
protected String wrapper (String s){  
    StringBuffer buf = new StringBuffer (s);  
    for (int i=0 ; i < s.length(); i+=40)  
        buf.insert(i + i/20 , "\n\t" );  
    return buf.toString();  
}  
}
```

6. Код примера. HTTPS сервер

Мы долго шли к этому примеру. И, наконец, я привожу код сервера с поддержкой SSL. Не правда ли, он действительно мал.

```
package arti.security;  
import java.io.*;  
import java.net.*;  
import java.security.*;  
import javax.net.ssl.*;  
  
public class SSLServer extends AbstractServer {  
  
    // дополнительно данному классу сервера  
    // потребуется иметь информацию о том  
    // где хранятся сертификаты, т.е. где находится keystore  
    // сам объект хранилища  
    KeyStore ks ;  
    // имя файла для хранения ключей и сертификатов  
    String fStoreName;  
    String keystorePassword , aliasPassword;  
    // признак того должен ли клиент сам представляться  
    //или же мы будем доверять всем входящим соединениям  
    boolean isClientSecure;  
  
    public SSLServer (String fName) throws IOException{  
        super (fName);  
    }  
}
```

Практическая криптография в Java. Создаем HTTPS-сервер

```
public boolean prepareForRun(){
    try {
        if (! super.prepareForRun()) return false;
        // ошибка возникла на стадии создания просто http сервера,
        // что уже говорить о защищенном сервере
        ks = KeyStore.getInstance(KeyStore.getDefaultType());
        fStoreName = XTools.getXPathText("HTTPd/SSL-Section/Keystore-path",
                                         configDoc.getDocumentElement());
        keystorePassword = XTools.getXPathText("HTTPd/SSL-Section/Keystore-password",
                                               configDoc.getDocumentElement());
        aliasPassword = XTools.getXPathText("HTTPd/SSL-Section/Alias-password",
                                             configDoc.getDocumentElement());

        isClientSecure = XTools.getElementByXPath(
            "HTTPd/SSL-Section/Need-client-authentication",
            configDoc.getDocumentElement()) != null;

        if (keystorePassword == null) {
            System.out.println("Пароль необходимый для доступа к
                хранилищу ключей не был найден в файле
                конфигурации, необходимо ввести его с клавиатуры");
            keystorePassword = new DataInputStream(System.in).readLine();
        }

        if (aliasPassword == null) {
            System.out.println("Пароль необходимый для доступа к
                ключу шифрования не был найден в файле
                конфигурации, необходимо ввести его с клавиатуры");
            aliasPassword = new DataInputStream(System.in).readLine();
        }

        ks.load(new FileInputStream(fStoreName), keystorePassword.toCharArray());
    }
    catch (Exception ex) {
        ex.printStackTrace();
        return false;
    }
    return true;
}
```

Практическая криптография в Java. Создаем HTTPS-сервер

```
// нам необходимо переписать данную функцию так,  
// чтобы на самом деле она возвращала  
// не объект типа ServerSocket , а сокет с поддержкой SSL  
// вся хитрость в том, что весь код  
// по обработке поступившего запроса совершенно не зависит  
// от того по какому именно протоколу был сделан запрос,  
// дело в том, что протокол HTTP  
// является протоколом более высокого уровня чем SSL,  
// следовательно прикладные приложения мало зависят от  
// того работают ли они по защищенному протоколу или нет  
protected ServerSocket createCoreSocket () throws IOException {  
  
    try {  
  
        /*printArray("Список набора протоколов алгоритмов  
            шифрования входящих в скиту по умолчанию",  
            fa.getDefaultCipherSuites());*/  
        KeyManagerFactory kmfa =  
            KeyManagerFactory.getInstance("SunX509");  
        // Создаем фабрику менеджеров ключей  
        //для генерации на основе их ключей  
        kmfa.init(ks, aliasPassword.toCharArray());  
  
        SSLContext sslco = SSLContext.getInstance("SSLv3");  
  
        // Привязать фабрику ключей к  
        //созданному контексту соединений по SSL3  
        sslco.init(kmfa.getKeyManagers(), null, null);  
  
        javax.net.ssl.SSLServerSocketFactory ssfa =  
            sslco.getServerSocketFactory();  
        // а теперь имя полностью готовую для работы  
        // фабрику ssl сокетов создадим и сам сокет  
        SSLServerSocket serverSocket =  
            (SSLServerSocket) ssfa.createServerSocket(localPort);  
        printArray ("Список разрешенных для использования  
            криптографических скит", serverSocket.getEnabledCipherSuites());  
        printArray ( "Список разрешенных для использования  
            протоколов", serverSocket.getEnabledProtocols());  
        serverSocket.setNeedClientAuth(isClientSecure);
```

```
        return serverSocket;
    }
    catch (Exception ex) {
        ex.printStackTrace();
        throw new IOException (
            "Ошибка на стадии создания SSL сокета: "+ ex);
    }
}

// данная функция служит для отображения окна сообщения с параметрами
// содержащими описания разрешенных к использованию скит протоколов
protected void printArray (String s , String sa[]){
    StringBuffer sb = new StringBuffer ();
    sb.append("-----" +s + "-----\n");
    for (int i=0; i < sa.length; i++)
        sb.append(sa[i]+\n");
    javax.swing.JOptionPane.showMessageDialog(null ,
        sb, s , javax.swing.JOptionPane.INFORMATION_MESSAGE);
}
}
```

7. Анализ кода. HTTPS сервер.

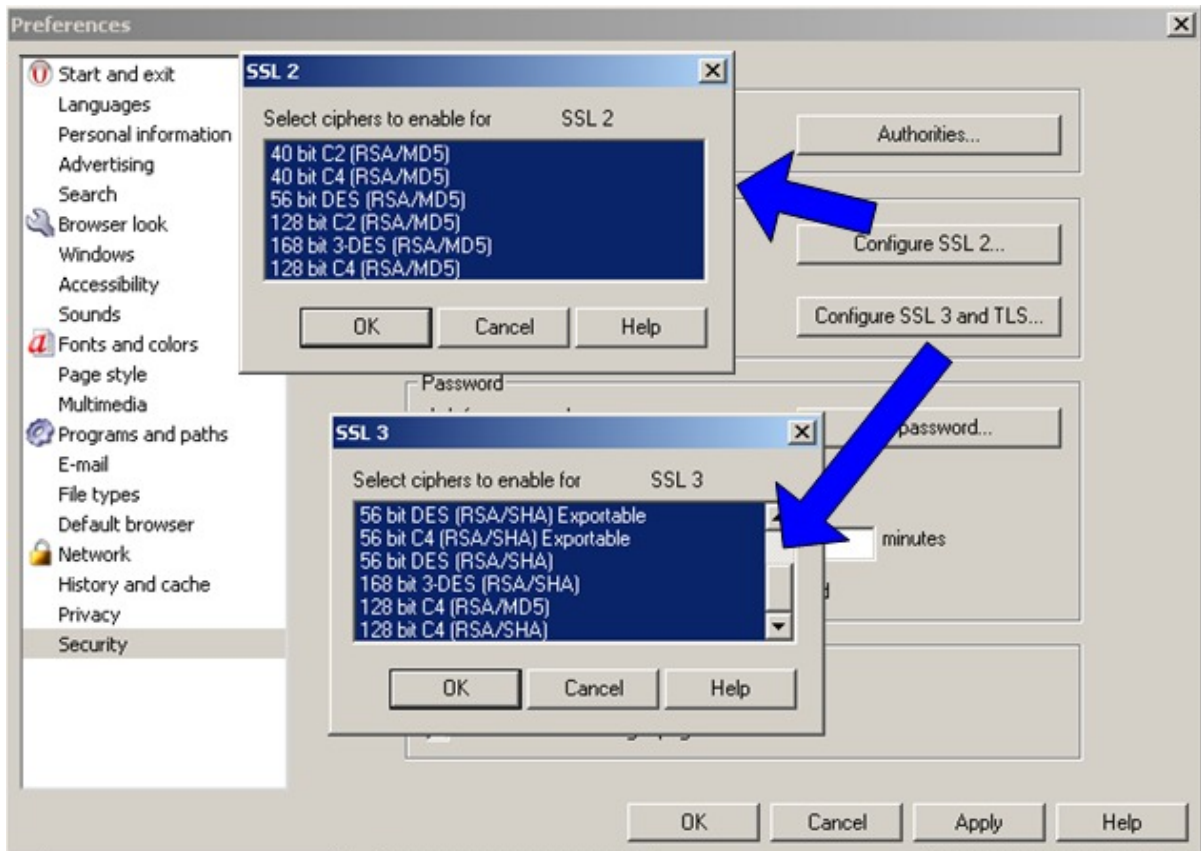
Для того чтобы запустить данный пример необходимо будет внести изменения в файл конфигурации. Я добавил в него набор элементов отвечающих за местоположение файла с сертификатами и ключами, т.е. файла используемого для чтения KeyStore. А также я добавил пароли для доступа к данному файлу и закрытым ключам внутри его. Разумеется, что это не правильный поступок, что же пожалуй будем считать что я дал вам домашнее задание исправить эту ошибку.

```
<SSL-Section>
  <Keystore-path>E:\mdocs\kolya\jbuilder_projects\HTTTPd-ssl\store
</Keystore-path>
  <Keystore-password>bigsecret</Keystore-password>
  <Alias-password>bigsecret</Alias-password>
  <Not-need-client-authentication />
</SSL-Section>
```

Если внимательно рассмотреть код класса SSLServer, то можно увидеть что изменения

были внесены в метод `prepareForRun`, этот метод теперь должен уметь выполнять дополнительное чтение из файла конфигурации информации о местоположении файла хранилища ключей, имени под которым закрытый ключ сервера хранится в файле-хранилище, пароле. И естественно что основной объем изменений был внесен в метод `createCoreSocket`. Теперь в данном методе создается экземпляр класса `SSLContext`, который инициализируется с помощью менеджеров ключей, создаваемых с помощью фабрики `KeyManagerFactory`. Затем создается фабрика серверных сокетов с поддержкой SSL, на этот раз мы пользуемся классом `SSLServerSocketFactory`. И последний шаг - это создание класса `SSLServerSocket`, который и возвращается как результат работы данного метода. По ходу выполнения на экран выводится информация о том какие именно криптографические сюиты поддерживаются созданным нами ssl-сокетом. Для этого используются методы `getEnabledCipherSuites` и `getEnabledProtocols`. Очевидно, что для того чтобы клиент и сервер могли договориться и поверить друг другу необходимо совпадение этих наборов. Ниже я привожу пример копии экрана браузера `opera`, с открытым диалоговым окном указания разрешенных к использованию сюит.

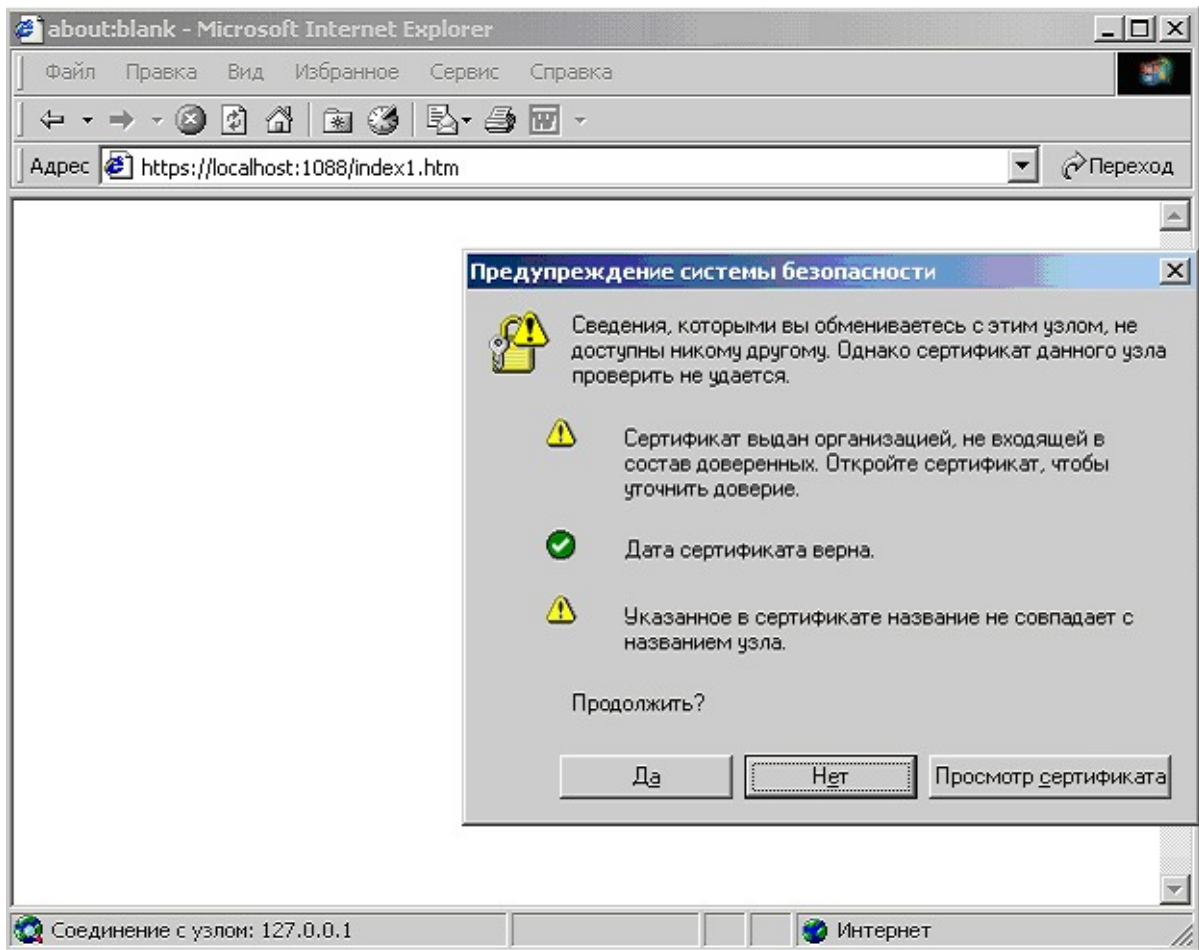
Практическая криптография в Java. Создаем HTTPS-сервер



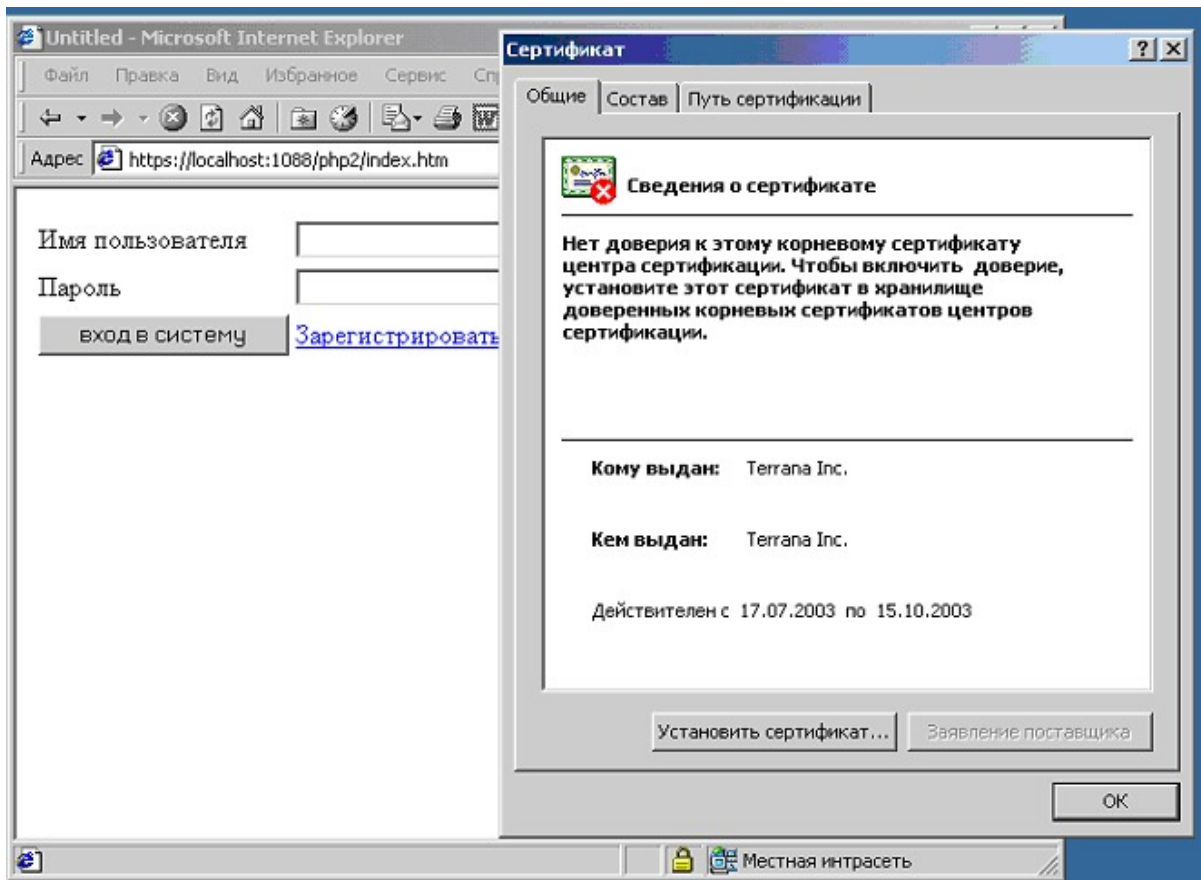
А теперь, внимание, затаим дыхание и попробуем запустить наш сервер и попробуем проверить как он работает с помощью любого браузера. Обратите внимание на то, что я указал в адресной строке браузера, я запрашиваю документ не по протоколу HTTP, а именно по HTTPS. В общем случае при работе в интернет принято соглашение, о том какие порты используются по умолчанию в тех или иных ситуациях. Так для протокола HTTP по умолчанию принят порт номер 80, а для HTTPS — 443. В данном частном случае я явно указал номер порта, соответствующий тому который был размещен в файле конфигурации.

При открытии страницы с сервера браузер говорит о том, что далее мы будем общаться с удаленной (вообще-то локальной т.к. я тестировал все это на одной машине, но смею вас уверить в том, что разницы не будет никакой) по защищенному протоколу, только вот системе неизвестен сертификат который предоставляет сервер. Так что мы должны его просмотреть и решить стоит ли нам доверять данной

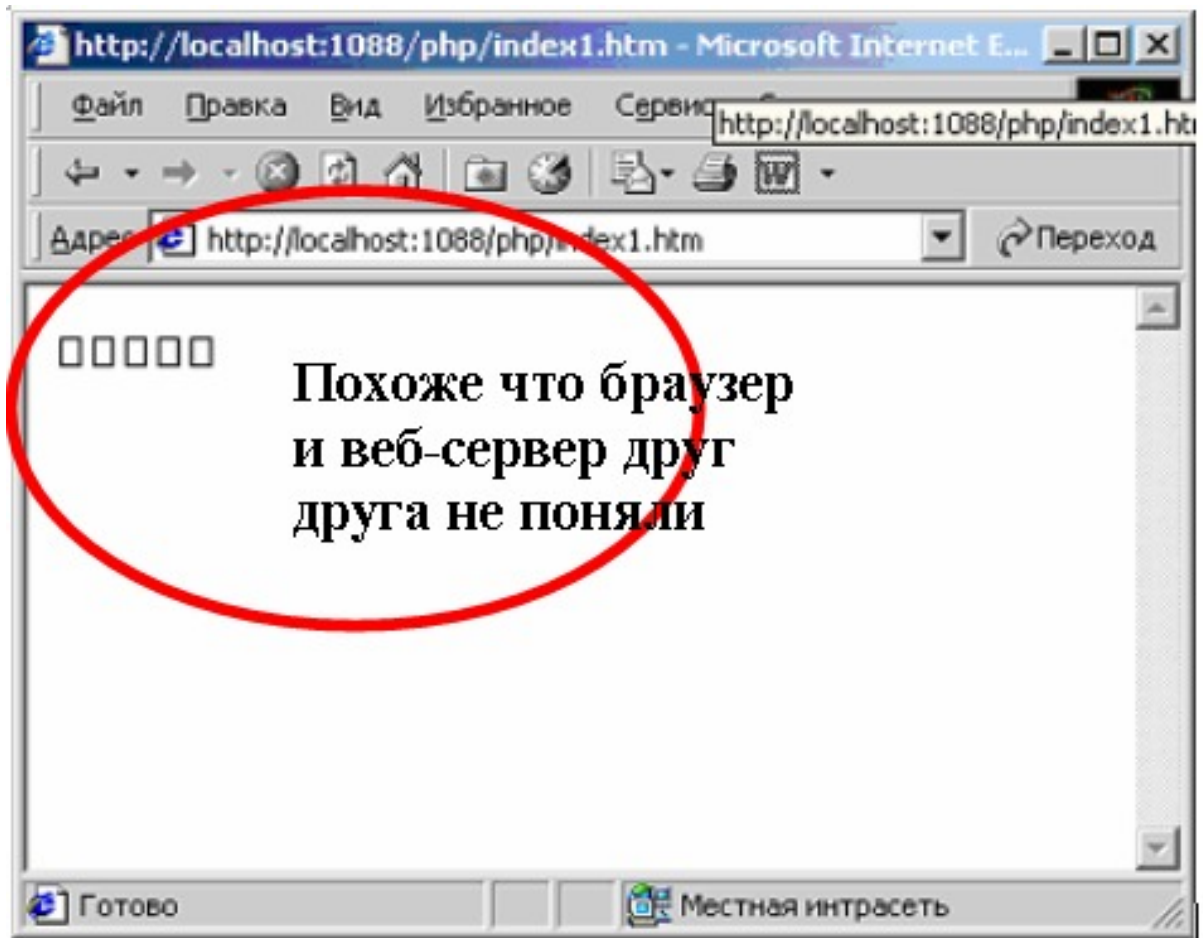
организации или нет. После того как мы скажем, что доверяем, при последующих запросах страниц с этого ресурса сообщение не будет выводиться. Но это будет только до тех пор пока не будет завершена сессия, т.е. более простым языком если вы закроете браузер при повторном обращении к ресурсу вас снова спросят на предмет того, доверяете вы такой организации как "Тетрана" или нет. Если вы уверены в том, что это хороший сайт и там нет злых вирусов и "троянов", то можно сертификат установить в локальное хранилище.



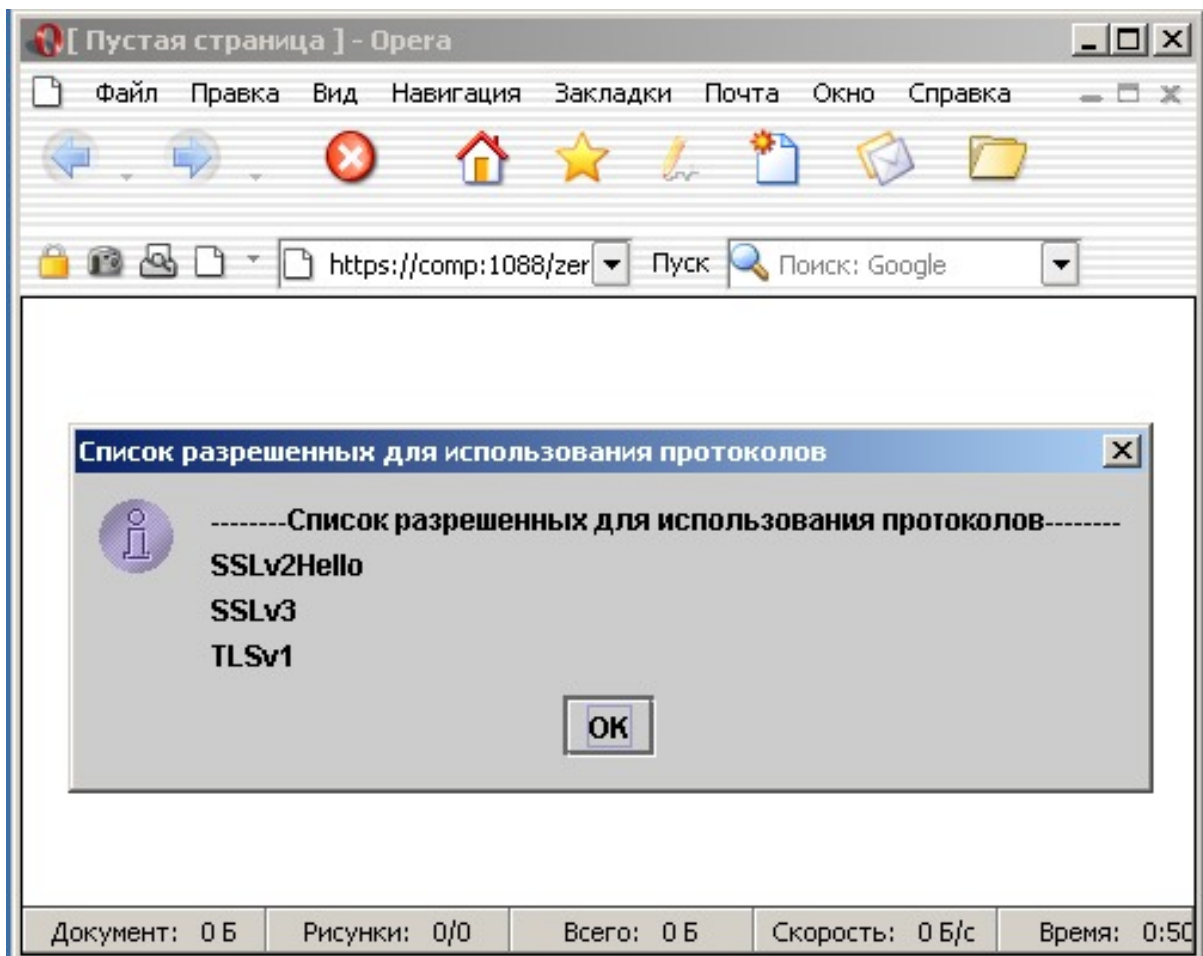
Практическая криптография в Java. Создаем HTTPS-сервер

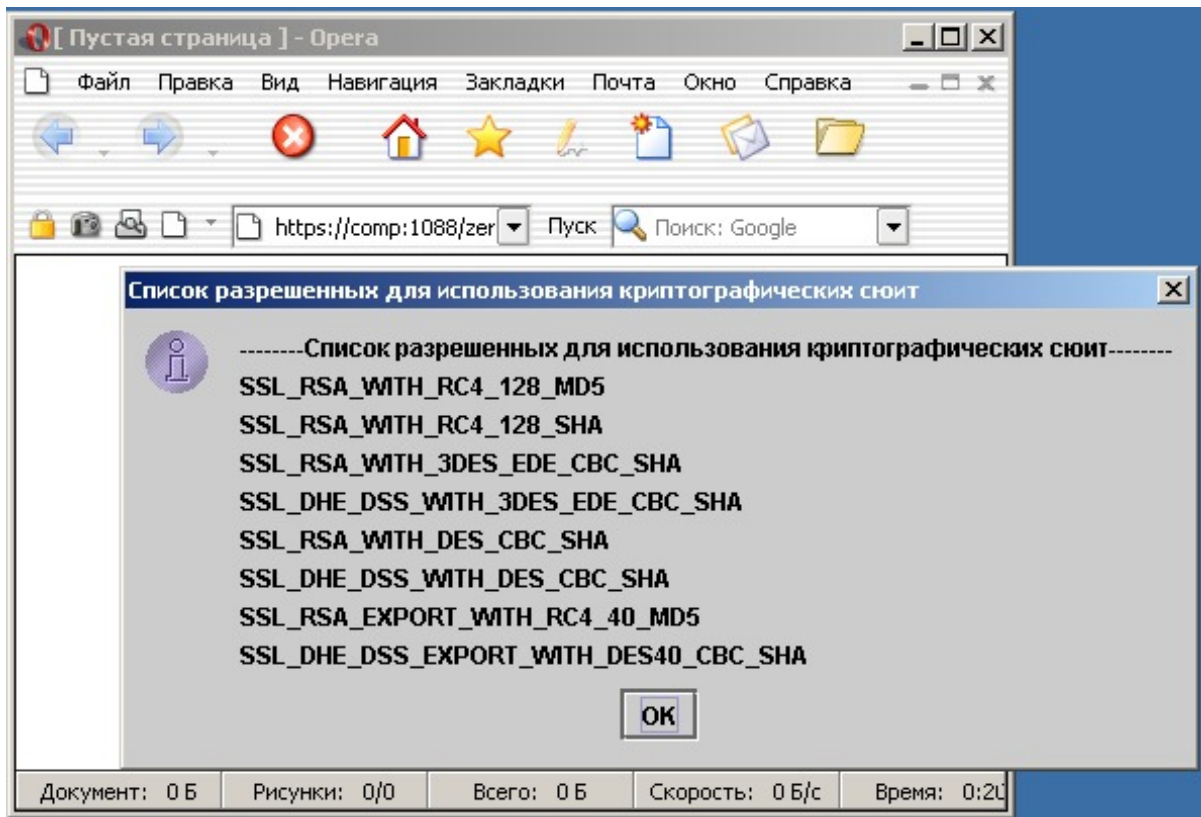


Кстати, я упоминал о том, что при обращении к защищенным ресурсам всегда следует указывать префикс протокола HTTPS, иначе вы получите у себя на экране нечто подобное этому.



Практическая криптография в Java. Создаем HTTPS-сервер





А вот и еще пара картинок с сообщениями которые выдал веб-сервер когда к нему подключился клиент, в этих сообщениях идет перечисление допустимых съюит криптографических расширений.

В самом начале этой серии статей я уже упоминал о том, какие существуют протоколы и чем они отличаются. Рекомендую на всякий случай обновить знания. А пока будем считать что пример закончен.

8. Есть ли шанс создать действительно надежную систему?

Самый неприятный момент в том, что огромное количество усилий, которые мы затратили, не могут в полной мере гарантировать того, что информация, которой будут обмениваться клиент и сервер, будет действительно надежно скрыта. В самом начале

данного цикла я упоминал отличия безусловно стойких алгоритмов и вычислительно стойких алгоритмов. Да, безусловно стойкие алгоритмы есть, они действительно дают строго математическое обоснование того, что даже обладание бесконечными вычислительными ресурсами не может дать злоумышленнику ни единого шанса на раскрытие нашей конфиденциальной информации. Увы, практическое применение данных алгоритмов невозможно, ибо возникают сложности, разрешение которых равноценно по затрате сил и энергии поиску этого самого абсолютно надежного способа шифрования.

В общем случае направления атаки на SSL наиболее рационально проводить именно на фронте человеческого, не даром социальная инженерия стала необходимым для любого уважающего хакера, зачем сидеть ночи напролет над ПК и искать дыры в реализации SSL для конкретного сервера или дыры в самом SSL (его математическое обоснование), можно просто позвонить и представившись кем посOLIDнее (например, директором организации) потребовать в ультимативной манере все пароли и прочее, обосновывая тем что, вчера был праздник и после него руководитель не может загрузить свой компьютер, потому что не помнит пароль. И так далее, я думаю, что ваша фантазия подскажет много других способов, как разрушить самый ненадежный компонент любой системы защиты — пользователя. Однако, данный материал не задумывался как практическое пособие по психологии и НЛП поэтому сосредоточимся именно на технических путях.

Путь первый наиболее простой — пойдем ломать в лоб. Метод перебора. В основе данного способа лежит очень простой и закономерный факт. Любой алгоритм шифрования зависит от небольшого кусочка секретной информации в общем случае который может быть представлен в виде последовательности битов, вроде "010110001101" и т.д. Очевидно, что раз длина подобного ключа конечна то ничего не мешает просто "тупо" перебрать все возможные варианты и попробовать каждый из них — подойдет или нет. Конечно, это займет много времени, но это уже другая забота. Кстати, увеличение количества битов длины ключа всего лишь на один приводит к увеличению затрат времени в два раза, а то что ряд 1, 2, 4, 8, ..., 2^n растет очень быстро это известный факт, вспомните замечательную историю про изобретателя шахмат и награду которую он затребовал "на первую клеточку положи 1 зернышко, на вторую — 2 зернышка, на третью ...", думаю, вспомнили, также как и то, что у бедного шаха (раджи) не хватило всех наличных средств, да и, пожалуй, если

взять все зерно, которое было выращено во всем мире за последние несколько десятилетий может не хватить. Поэтому если установить длину ключа равной тем же 64 битам, то можно спокойно спать и не думать о том, что завтра ваш шифр будет открыт. Почти. Компьютеры то же ведь работают быстро, и с каждым годом все быстрее. И даже если у злоумышленника под кроватью не валяется пыльный стау, то существует еще несколько способов ускорить подбор ключа. Я для иллюстрации привожу табличку которую вытянул с <http://www.ssl.stu.neva.ru/psw/crypto/keylen.doc> В данной таблице идет оценка нижних границ затрат времени и сил для взлома различными представителями преступного и делового мира.

Тип нападающего	Бюджет	Инструмент	Время и цена за взломанный ключ (40 бит)	Время и цена за взломанный ключ (56 бит)	Длина необходимая для защиты
Пеший хакер	малый	мусорное машинное время	1 неделя	невозможно	45
	400 долл.	FPGA	5 часов (\$0.08)	38 лет (\$5,000)	50
Малый бизнес	10 тыс.долл	FPGA	12 минут (\$0.08)	556 дней (\$5,000)	55
Средний бизнес	300 тыс. долл	FPGA or ASIC	24 секунды (\$0.08) .18 секунд (\$0.001)	19 дней (\$5,000) 3 часа (\$38)	60
Крупная компания	10 млн. долл.	FPGA или ASIC	0.7 секунд (\$0.08) 0,005 секунд (\$0.001)	13 часов (\$5,000) .6 минут (\$38)	70
Спецслужбы	300 млн. долл.	ASIC	0.0002 секунд (\$0.001)	12 секунд (\$38)	75

Итак, один стау нам не поможет и это очевидно, как поступить? Конечно, объединить много компьютеров в единую распределенную сеть. Подобный подход не нов и применяется не только для взлома шифров, надеюсь все слышали и может быть участвовали в всемирной программе поиска разума иных межзвездных миров, поиске лекарств от разных опасных болезней или расшифровывали геном человека. В общем случае идея такова: у вас на компьютере должна быть запущена специальная

программа, которая получает из сети интернет пакеты с заданиями, выполняет их обработку в моменты простоя компьютера, попутно демонстрируя вам красивые заставки, по завершению отсылает блок обработанной информации на центральный сервер, а если повезет и именно ваш компьютер вычислит искомую информацию, то вам подарят кофеварку и песочные часы. Взломом криптографических алгоритмов занимаются не только злые хакеры и конкурирующие компании, но и например, некоммерческий проект www.distributed.net давно уже пытается бороться с экспортными ограничениями на длину ключа крипто-приложений которые были определены правительством США. Белый дом свято верит, что длины ключа 40 бит, или 56 с верхом хватает на то, чтобы надежно защитить информацию от злоумышленников, не создав при этом никаких трудностей для доблестных сотрудников плаща и кинжала с целью обеспечения национальной безопасности и мира во всем мире, раскрыть секретные переговоры мировых террористов. В рамках проекта www.distributed.net уже успешно было доказано что применение ключей длиной 56 бит и менее не может быть надежной защитой. С точки зрения математики задача компрометации алгоритма RSA сводится к необходимости найти способ разложения большого числа на простые множители. И никто не может дать гарантии в том, что через несколько лет, месяцев, а может быть и завтра какой-нибудь гениальный математик не откроет данный алгоритм. Недавно, пришлось прочитать в сети следующую информацию:

Замечание:

Как известно в 1994 году был принят первый стандарт РФ в области ЭЦП — ГОСТ Р34.10-94 «Информационная технология. Криптографическая защита информации. Процедуры выработки и проверки электронной цифровой подписи на базе асимметричного криптографического алгоритма». Технически в данном стандарте был использован алгоритм Эль-Гамала, который был основан на сложности выполнения задачи дискретного логарифмирования в конечных полях, однако и увы, вскоре появился алгоритм метода решета числового поля. И как вы уже догадались, вскрытие криптосистемы на основе данного алгоритма уже не составляет сложности. С подобной точки зрения разработчики программных продуктов более заинтересованы в использовании давно известных алгоритмов, пусть даже они имеют низкий уровень затрат на открытие, но при этом есть гарантия что в ближайшее время не случится очередной прорыв, и все решения которые были созданы лопнут как мыльный пузырь.

Но это только первый путь, в любой хорошей книжке по криптографии, обычно излагается и перечисляется множество способов атаки на криптосистему. Основные пути атаки это:

1. Нападение на основе известного текста и криптотекста, в подобном случае шансы криптоаналитика на взлом ключа заметно повышаются, а подобные ситуации,

когда известно сообщение или его часть отнюдь не редки. Согласитесь сами, что в любом документе есть шапка, в общем случае шаблон, который повторяется из документа в документ и т.д.

2. Нападение на основании специально подготовленного криптоаналитиком противника текста и его шифртекста. Если противник имел возможность создать собственный текст и выполнить его шифрование вашим ключом, а потом смог его дешифровать, то у него явно повышаются шансы добиться раскрытия уже ваших секретов.
3. Посредник, — идея данной атаки в следующем: злоумышленник представляется законным пользователям криптосистемы Васе и Пете, Васе соответственно как Петя, а Пете — как Вася. Вася и Петя обмениваются информацией веря в то что никто ее не узнает, а на самом деле весь трафик идет через посредника-злоумышленника.
4. Способ повторения — как там говорится в древней китайской мудрости "если очень долго сидеть на берегу реки, то можно увидеть, как мимо проплывают трупы твоих врагов". Идея взлома примитивно проста — злоумышленник подключается к каналу связи, по которому обмениваются информацией законные пользователи криптосистемы и перехватывает передаваемые запросы. Например, если процедура аутентификации заключается в разделении общего секрета: сервер предлагает клиенту (предполагаемому, разумеется) некоторое число, а клиент должен выполнить над этим числом некоторое нетривиальное преобразование и вернуть серверу для сверки. Злоумышленник, достаточно долго наблюдая за сетью может составить специальную таблицу соответствий чисел и откликов на них, не имея при этом ни малейшего представления о том, какие именно преобразования выполняются на самом деле. И в следующий раз, когда сервер пошлет запрос и повторит число, то злоумышленник не упустит шанса и отправит сохраненное в таблице значение правильного отклика. Все. Связь установлена.

На этой торжественной ноте я заканчиваю текущую статью. Продолжение следует.

9. Заключение

Несмотря на то, что эта статья носила характер более практический чем теоретический, нам наконец удалось собрать воедино все знания которые я подводил в предыдущей статье и создать работающий прототип защищенного веб-сервера. Нам

также удалось добавить теоретические моменты связанные с описанием ряда классов и интерфейсов Java Crypto API, необходимых для создания сессий SSL. Также последняя часть этой статьи была посвящена вопросу оценки надежности используемых криптоалгоритмов.

10. Ресурсы

1. Раз уж мы начали создавать веб-сервер неплохо было бы получить более подробные сведения по протоколу HTTP, HTTPS, заголовкам протокола, вполне неплохая документация всегда лежит на сайте www.citforum.ru.
2. Читайте документацию на сайтах www.verisign.com, www.instantssl.com, www.thawte.com — там вы найдете подробную документацию по вопросам генерации сертификатов для различных коммерческих веб-серверов и процедуре их последующей инсталляции, кроме того, может быть, вам даже выдадут бесплатный сертификат на пару дней для пробы.
3. Очень приятный для чтения материал по защите и взлому <http://www.computerra.ru/offline/2003/487/25680/>.
4. www.ssl.stu.neva.ru — Специализированный центр защиты информации. Действительно много информации.