HTTPS工作流程

题目标签

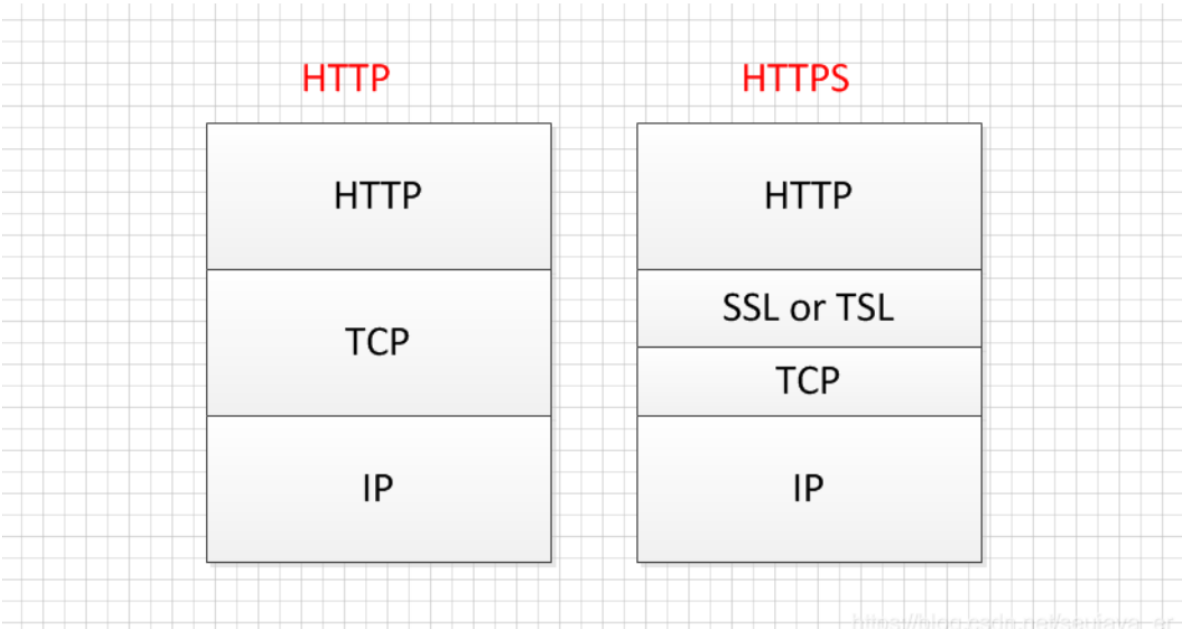
学习时常：20分钟

题目难度：中级

知识点标签：传输协议、加密算法

HTTPS的介绍

HTTPS是一种应用层协议，本质上来说它是HTTP协议的一种变种。HTTPS比HTTP协议安全，因为HTTP是明文传输，而HTTPS是加密传输，加密过程中使用了三种加密手段，分别是证书，对称加密和非对称加密。HTTPS相比于HTTP多了一层SSL/TSL，其构造如下：



简单介绍下加密算法

（1）证书加密

服务器在使用证书加密之前需要去证书颁发机构申请该服务器的证书，在HTTPS的请求过程服务器端将会把本服务器的证书发送给客户端，客户端进行证书验证，以此来验证服务器的身份。

（2）对称加密

HTTPS的请求中，客户端和服务器之间的通信的数据是通过对称加密算法进行加密的。对称加密，即在加密和解密的过程使用同一个私钥进行加密以及解密，而且对称加密算法是公开的，***所以该私钥是不能够泄漏的，一旦泄漏，对称加密形同虚设。***

上述私钥是可能泄漏的，原因是上述私钥是需要在网络中进行传输的。流程：在A端生成私钥，传递给B端（传递过程需要是安全的），后面A端使用该私钥加密，传递数据报文到B端，B端使用接受到的私钥解密。

加密过程：加密算法+明文+私钥------》密文

解密过程：解密算法+密文+私钥------》明文

适用场景：上述过程是不复杂的，对大量数据进行加密时，对称加密是适用的，速度快

（3）非对称加密算法

HTTPS的请求中也使用了非对称加密算法。非对称加密，加密和解密过程使用不同的密钥，一个公钥，对外公开，一个私钥，仅是解密端拥有。由于公钥和私钥是分开的，非对称加密算法安全级别高，加密密文长度有限制，适用于对少量数据进行加密，速度较慢。

case1：使用公钥加密，私钥解密

加密过程：加密算法+明文+共钥------》密文

解密过程：解密算法+密文+私钥------》明文

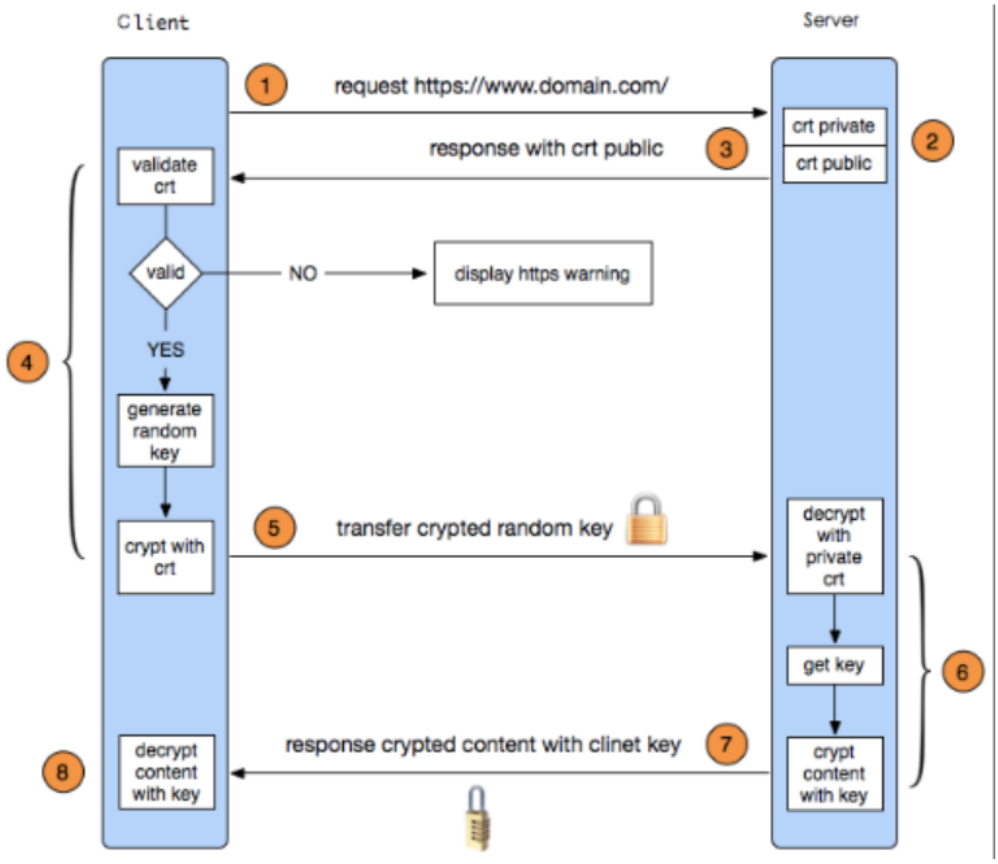
流程：A端向B端请求，B端返回公钥给A端，然后A端使用公钥加密，传递给数据报文给B端，B端使用自己的私钥进行解密。

case2：使用私钥加密，公钥解密

加密过程：加密算法+明文+私钥------》密文

解密过程：解密算法+密文+共钥------》明文

HTTPS进行的流程



1.客户端向服务端发送请求，客户端主要向服务器提供以下信息：

- 支持的协议版本，比如TLS 1.0版。
- 一个客户端生成的随机数，稍后用于生成"对话密钥"。
- 支持的加密方法，比如RSA公钥加密。
- 支持的压缩方法。

2.服务器端收到请求后，向客户端做出回应，回应的内容包括：

- 确认使用的加密通信协议版本，比如TLS 1.0版本。如果浏览器与服务器支持的版本不一致，服务器关闭加密通信。
- 一个服务器生成的随机数，稍后用于生成"对话密钥"。
- 确认使用的加密方法，比如RSA公钥加密。
- 服务器证书。

3.客户端收到服务器回应以后，首先验证服务器证书。

如果证书不是可信机构颁布、或者证书中的域名与实际域名不一致、或者证书已经过期，就会向访问者显示一个警告，如下

如果证书没有问题，客户端就会从证书中取出服务器的公钥。然后，向服务器发送下面三项信息:

- 一个随机数。该随机数用服务器公钥加密，防止被窃听。
- 编码改变通知，表示随后的信息都将用双方商定的加密方法和密钥发送。
- 客户端握手结束通知，表示客户端的握手阶段已经结束。这一项同时也是前面发送的所有内容的hash值，用来供服务器校验

证书的验证过程如下：

CA机构在签发证书的时候，都会使用自己的私钥对证书进行签名，如果我们使用的是购买的证书，那么很有可能，颁发这个证书的CA机构的公钥已经预置在操作系统中。这样浏览器就可以使用CA机构的公钥对服务器的证书进行验签，验签之后得到的是CA机构使用sha256得到的证书摘要，客户端就会对服务器发送过来的证书使用sha256进行哈希计算得到一份摘要，然后对比之前由CA得出来的摘要，就可以知道这个证书是不是正确的，是否被修改过。

4. 服务端回应

服务器收到客户端的第三个随机数之后，计算生成本次会话所用的"会话密钥"。然后，向客户端最后发送下面信息：

- 编码改变通知，表示随后的信息都将用双方商定的加密方法和密钥发送。
- 服务器握手结束通知，表示服务器的握手阶段已经结束。这一项同时也是前面发送的所有内容的hash值，用来供客户端校验。

会话秘钥是根据前面几次对话过程中产生的三个随机数以及一些其他算法产生的，后面服务器与客户端的交互都是通过这对话秘钥进行加密解密处理的，其他的都和HTTP协议一样。

HTTPS中自签名证书的生成

需要注意的是接下来几种文件的类型：

- key是服务器上的私钥文件，用于对发送给客户端数据的加密，以及对从客户端接收到数据的解密
- csr是证书签名请求文件（公钥），用于提交给证书颁发机构（CA）对证书签名

- crt是由证书颁发机构（CA）签名后的证书，或者是开发者自签名的证书，包含证书持有人的信息，持有人的公钥，以及签署者的签名等信息
- keystore 包含证书的文件，可以自己去导入证书
- PEM 文件格式存储证书和密钥，用于导出，导入证书时候的证书的格式，有证书开头，结尾的格式。

还有就是X.509是一个标准，规范了公开秘钥认证、证书吊销列表、授权凭证、凭证路径验证算法等。

服务器端用户证书的生成过程：

1. 生成私钥（.key）文件

```
# private key
$openssl genrsa -des3 -out server.key 1024
```

2. 生成证书请求（.csr）文件（公钥）

```
# generate csr
$openssl req -new -key server.key -out server.csr
```

Country Name (2 letter code) [XX]:CN---------- 证书持有者所在国家

State or Province Name (full name) []:BJ---------- 证书持有者所在州或省份（可省略不填）

Locality Name (eg, city) []:BJ---------- 证书持有者所在城市（可省略不填）

Organization Name (eg, company) []:NH---------- 证书持有者所属组织或公司

Organizational Unit Name (eg, section) []:.---------- 证书持有者所属部门（可省略不填）

Common Name (eg, your name or your server's hostname) []:www.hellobcdb.com----- **必须要填写域名或者ip地址**

Email Address []:---------- 邮箱（可省略不填）

challenge password:.....自定义密码

An optional company name：.....（可选）公司名称

3. 自签名的证书文件

```
openssl x509 -req -days 365 -in server.csr -signkey server.key -out server.crt
```

4.本案例需要用到pem格式的证书，可以用以下方式合并证书文件（crt）和私钥文件（key）来生成

```
cat server.crt server.key > server.pem
```

客户端证书文件的生成

这里采用的是自签名的方式，所以客户端需要有一个导入服务端证书的文件，以供客户端去验证服务段的证书过程。

1.生成.keystore文件

```
keytool -genkey -validity 36000 -alias www.hellobcdb.com -keyalg RSA -keystore
client.keystore
```

2.导入服务端证书

```
keytool -import -alias serverkey -file server.crt -keystore client.keystore
```

网上案例：

环境及工具：ubuntu16.04，QtCreator，mongoose(cesanta)，java环境。

服务端代码：

```
#include "mongoose.h"

static const char *s_http_port = "8443";
static const char *s_ssl_cert = "/home/gqx/workplace/TestHttps/server.pem";
static const char *s_ssl_key = "/home/gqx/workplace/TestHttps/server.key";
static struct mg_serve_http_opts s_http_server_opts;

static void ev_handler(struct mg_connection *nc, int ev, void *p) {
  if (ev == MG_EV_HTTP_REQUEST) {
    mg_serve_http(nc, (struct http_message *) p, s_http_server_opts);
  }
}

int main(void) {
  struct mg_mgr mgr;
  struct mg_connection *nc;
  struct mg_bind_opts bind_opts;
  const char *err;

  mg_mgr_init(&mgr, NULL);
  memset(&bind_opts, 0, sizeof(bind_opts));
  bind_opts.ssl_cert = s_ssl_cert;
  bind_opts.ssl_key = s_ssl_key;
  bind_opts.error_string = &err;

  printf("Starting SSL= server on port %s, cert from %s, key from %s\n",
         s_http_port, bind_opts.ssl_cert, bind_opts.ssl_key);
  nc = mg_bind_opt(&mgr, s_http_port, ev_handler, bind_opts);
  if (nc == NULL) {
    printf("Failed to create listener: %s\n", err);
    return 1;
  }

  // Set up HTTP server parameters
  mg_set_protocol_http_websocket(nc);
```

```
  s_http_server_opts.document_root = ".";  // Serve current directory
  s_http_server_opts.enable_directory_listing = "yes";

  for (;;) {
    mg_mgr_poll(&mgr, 1000);
  }
  mg_mgr_free(&mgr);

  return 0;
}
```

QT中C++项目的pro文件内容如下，注意要添加相应的库：

```
TEMPLATE = app
CONFIG += console c++11
CONFIG -= app_bundle
CONFIG -= qt
LIBS += -lpthread
LIBS += -lssl -lcrypto
LIBS += -L /usr/local/bin -lcryptopp -ldl -lz
SOURCES += main.cpp \
    mongoose.cpp
HEADERS += \
    mongoose.h
```

java客户端代码

HttpsClient类

```
package com.gqx.test;

import java.io.FileInputStream;
import java.io.IOException;
import java.net.Socket;
import java.net.UnknownHostException;
import java.security.KeyManagementException;
import java.security.KeyStore;
import java.security.KeyStoreException;
import java.security.NoSuchAlgorithmException;
import java.security.UnrecoverableKeyException;
import java.security.cert.CertificateException;
import java.security.cert.X509Certificate;

import javax.net.ssl.SSLContext;
import javax.net.ssl.TrustManager;
import javax.net.ssl.X509TrustManager;

import org.apache.http.HttpVersion;
import org.apache.http.client.HttpClient;
import org.apache.http.conn.ClientConnectionManager;
import org.apache.http.conn.scheme.PlainSocketFactory;
import org.apache.http.conn.scheme.Scheme;
import org.apache.http.conn.scheme.SchemeRegistry;
import org.apache.http.conn.ssl.SSLSocketFactory;
import org.apache.http.impl.client.DefaultHttpClient;
import org.apache.http.impl.conn.tsccm.ThreadSafeClientConnManager;
import org.apache.http.params.BasicHttpParams;
```

```
import org.apache.http.params.HttpConnectionParams;
import org.apache.http.params.HttpParams;
import org.apache.http.params.HttpProtocolParams;
import org.apache.http.protocol.HTTP;

public class HttpsClient {

    private static final int SET_CONNECTION_TIMEOUT = 5 * 1000;
    private static final int SET_SOCKET_TIMEOUT = 20 * 1000;

    public static HttpClient getNewHttpClient() {
        try {
            FileInputStream fis = new FileInputStream("/home/gqx/文
档/oscar/client.keystore");
            String storepass = "starchain";

            KeyStore trustStore =
KeyStore.getInstance(KeyStore.getDefaultType());
            trustStore.load(fis, storepass.toCharArray());

            SSLSocketFactory sf = new MySSLSocketFactory(trustStore);

sf.setHostnameVerifier(SSLSocketFactory.ALLOW_ALL_HOSTNAME_VERIFIER);

            HttpParams params = new BasicHttpParams();

            HttpConnectionParams.setConnectionTimeout(params, 10000);
            HttpConnectionParams.setSoTimeout(params, 10000);

            HttpProtocolParams.setVersion(params, HttpVersion.HTTP_1_1);
            HttpProtocolParams.setContentCharset(params, HTTP.UTF_8);

            SchemeRegistry registry = new SchemeRegistry();
            registry.register(new Scheme("http",
PlainSocketFactory.getSocketFactory(), 80));
            registry.register(new Scheme("https", sf, 443));

            ClientConnectionManager ccm = new
ThreadSafeClientConnManager(params, registry);

            HttpConnectionParams.setConnectionTimeout(params,
SET_CONNECTION_TIMEOUT);
            HttpConnectionParams.setSoTimeout(params, SET_SOCKET_TIMEOUT);
            DefaultHttpClient client = new DefaultHttpClient(ccm, params);

            return client;
        } catch (Exception e) {
            return new DefaultHttpClient();
        }
    }




    private static class MySSLSocketFactory extends SSLSocketFactory {
        SSLContext sslContext = SSLContext.getInstance("TLS");
```

```
        public MySSLSocketFactory(KeyStore truststore) throws
NoSuchAlgorithmException,
                KeyManagementException, KeyStoreException,
UnrecoverableKeyException {
            super(truststore);

            TrustManager tm = new X509TrustManager() {
                public void checkClientTrusted(X509Certificate[] chain, String
authType)
                        throws CertificateException {
                }

                public void checkServerTrusted(X509Certificate[] chain, String
authType)
                        throws CertificateException {
                }

                public X509Certificate[] getAcceptedIssuers() {
                    return null;
                }
            };

            sslContext.init(null, new TrustManager[] { tm }, null);
        }

        @Override
        public Socket createSocket(Socket socket, String host, int port, boolean
autoClose)
                throws IOException, UnknownHostException {
            return sslContext.getSocketFactory().createSocket(socket, host,
port, autoClose);
        }

        @Override
        public Socket createSocket() throws IOException {
            return sslContext.getSocketFactory().createSocket();
        }
    }
}
```

测试类，post请求

```
package com.gqx.test;

import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;
import java.util.Set;
import java.util.stream.Collectors;
import org.apache.http.HttpEntity;
import org.apache.http.HttpResponse;
import org.apache.http.client.HttpClient;
import org.apache.http.client.entity.UrlEncodedFormEntity;
import org.apache.http.client.methods.HttpPost;
```

```
import org.apache.http.message.BasicNameValuePair;

public class TestMain {

    public static void main(String[] args) {
        String urlStr = "https://www.hellobcdb.com:7999/fff";
        Map<String,String> params = new HashMap<>();
        params.put("value", "publish.1666,'Database','computer
science','Alice'");
        params.put("name", "gqx");
        params.put("password", "111");
        try {
            //DefaultHttpClient client = new DefaultHttpClient();
            HttpClient client = HttpsClient.getNewHttpClient();
            HttpPost httpPost = new HttpPost(urlStr);

            HttpEntity entity;

            ArrayList<BasicNameValuePair> pairs = new
ArrayList<BasicNameValuePair>();
            if(params != null){
                Set<String> keys = params.keySet();
                for(Iterator<String> i = keys.iterator(); i.hasNext();) {
                    String key = (String)i.next();
                    pairs.add(new BasicNameValuePair(key, params.get(key)));
                }
            }
            entity = new UrlEncodedFormEntity(pairs, "utf-8");
            httpPost.setEntity(entity);
            //Log.i(TAG, "post总字节数:"+entity.getContentLength());
            HttpResponse response = client.execute(httpPost);
            try {
                // 获取响应实体
                HttpEntity entitys = response.getEntity();
                System.out.println("--------------------------------------");
                // 打印响应状态
                System.out.println(response.getStatusLine());
                if (entitys != null) {
                    // 打印响应内容长度
                    String result = new BufferedReader(new
InputStreamReader(entitys.getContent()))

.lines().collect(Collectors.joining(System.lineSeparator()));
                       System.out.println(result);
                }
                System.out.println("--------------------------------------");
            } catch (Exception e) {
                e.printStackTrace();
            }
        }catch(Exception e){
            e.printStackTrace();
        }
    }
}
```

相关jar包下载 ： https://files.cnblogs.com/files/helloworldcode/lib.zip

HTTPS和HTTP的区别

（1）HTTPS是密文传输，HTTP是明文传输；

（2）默认连接的端口号是不同的，HTTPS是443端口，而HTTP是80端口；

（3）HTTPS请求的过程需要CA证书要验证身份以保证客户端请求到服务器端之后，传回的响应是来自于服务器端，而HTTP则不需要CA证书；

（4）HTTPS=HTTP+加密+认证+完整性保护。