<http://blog.sina.com.cn/s/blog_591c7a8f0102vdoa.html>

**互联网全站HTTPS的时代已经到来**

by @BAIDU罗成

**前言**

       我目前在百度从事HTTPS方面的性能优化工作。百度无线搜索目前已经支持https，手机访问地址是https://m.baidu.com。

       在HTTPS项目的开展过程中明显感觉到目前国内互联网对HTTPS并不是很重视，其实也就是对用户隐私和网络安全不重视。本文从保护用户隐私的角度出发，简单描述现在存在的用户隐私泄露和流量劫持现象，然后进一步说明为什么HTTPS能够保护用户安全以及HTTPS使用过程中需要注意的地方。

       国外很多网站包括google,facebook,twitter都支持了全站HTTPS，而国内目前还没有一家大型网站全站支持HTTPS（PC端的微信全部使用了HTTPS，但是PC端用户应该不多）。甚至一些大型网站明显存在很多HTTPS使用不规范或者过时的地方。比如支付宝使用的是tls1.0和RC4,而京东(quickpay.jd.com)竟然还使用着SSL3.0这个早就不安全并且性能低下的协议，其他很多网站的HTTPS登陆页面也存在着不安全的HTTP链接，这个也为黑客提供了可乘之机。

       由于篇幅关系，文中几乎没有详细描述任何细节，后面有时间我再一一整理成博客发表出来。同时由于水平有限，本文肯定存在很多错误，希望大家不吝赐教。本文的大部分内容都能从互联网上搜索到，有些地方我也标明了引用，可以直接跳转过去。但是全文都是在我自己理解的基础上结合开发部署过程中的一些经验和测试数据一个字一个字敲出来的，最后决定将它们分享出来的原因是希望能和大家多多交流，抛砖引玉，共同推进中国互联网的HTTPS发展。

本文不会科普介绍HTTPS、TLS及PKI，如果遇到一些基本概念文中只是提及而没有描述，请大家自行百度和google。本文重点是想告诉大家HTTPS没有想像中难用可怕，只是没有经过优化。

       中国互联网全站使用HTTPS的时代已经到来。

**1，用户隐私泄露的风险很大**

       人们的生活现在已经越来越离不开互联网，不管是社交、购物还是搜索，互联网都能带给人们很多的便捷。与此同时，用户“裸露”在互联网的信息也越来越多，另一个问题也日益严重，那就是隐私和安全。

        几乎所有的互联网公司都存在用户隐私泄露和流量劫持的风险。BAT树大招风，这方面的问题尤其严重。比如用户在百度搜索一个关键词，“人流”，很快就会有医院打电话过来推销人流手术广告，不知情的用户还以为是百度出卖了他的手机号和搜索信息。同样地，用户在淘宝搜索的关键词也很容易被第三方截获并私下通过电话或者其他广告形式骚扰用户。而QQ和微信呢，显然用户不希望自己的聊天内容被其他人轻易知道。为什么BAT不可能出卖用户隐私信息给第三方呢？因为保护用户隐私是任何一个想要长期发展的互联网公司的安身立命之本，如果用户发现使用一个公司的产品存在严重的隐私泄露问题，显然不会再信任该公司的产品，最终该公司也会因为用户大量流失而陷入危机。所以任何一家大型互联网公司都不可能因为短期利益而出卖甚至忽视用户隐私。

       那既然互联网公司都知道用户隐私的重要性，是不是用户隐私就得到了很好的保护呢？现实却并不尽如人意。由于目前的WEB应用和网站绝大部分是基于HTTP协议，国内没有任何一家大型互联网公司采用全站HTTPS方案来保护用户隐私（排除支付和登陆相关的网站或者页面以及PC端的微信）。因为HTTP协议简单方便，易于部署，并且设计之初也没有考虑安全性，所有内容都是明文传输，也就为现在的安全问题埋下了隐患。用户在基于HTTP协议的WEB应用上的传输内容都可以被中间者轻易查看和修改。

      比如你在百度搜索了一个关键词“https“，中间者通过tcpdump或者wireshark等工具就很容易知道发送请求的全部内容。wireshark的截图如下：

[](http://photo.blog.sina.com.cn/showpic.html#blogid=591c7a8f0102vdoa&url=http://album.sina.com.cn/pic/001Db1PVzy6N1nKzopO48)

     　这里所谓的中间者是指网络传输内容需要经过的网络节点，既有硬件也有软件，比如中间代理服务器、路由器、小区WIFI热点、公司统一网关出口等。这里面最容易拿到用户内容的就是各种通信服务运营商和二级网络带宽提供商。而最有可能被第三方黑客动手脚的就是离用户相对较近的节点。

        中间者为什么要查看或者修改用户真实请求内容呢？很简单，为了利益。常见的几种危害比较大的中间内容劫持形式如下：

    获取无线用户的手机号和搜索内容并私下通过电话广告骚扰用户。为什么能够获取用户手机号？呵呵，因为跟运营商有合作。

    获取用户帐号cookie，盗取帐号有用信息。

    在用户目的网站返回的内容里添加第三方内容，比如广告、钓鱼链接、植入木马等。

    　总结来讲，由于HTTP明文传输，同时中间内容劫持的利益巨大，所以用户隐私泄露的风险非常高。

**2，HTTPS能有效保护用户隐私**

       HTTPS就等于HTTP加上TLS（SSL)，HTTPS协议的目标主要有三个：

    数据保密性。保证内容在传输过程中不会被第三方查看到。就像快递员传递包裹时都进行了封装，别人无法知道里面装了什么东西。

    数据完整性。及时发现被第三方篡改的传输内容。就像快递员虽然不知道包裹里装了什么东西，但他有可能中途掉包，数据完整性就是指如果被掉包，我们能轻松发现并拒收。

    身份校验。保证数据到达用户期望的目的地。就像我们邮寄包裹时，虽然是一个封装好的未掉包的包裹，但必须确定这个包裹不会送错地方。

      通俗地描述上述三个目标就是封装加密，防篡改掉包，防止身份冒充，那TLS是如何做到上述三点的呢？我分别简述一下。

2.1 数据保密性

2.1.1 非对称加密及密钥交换

        数据的保密性主要是通过加密完成的。加密算法一般分为两种，一种是非对称加密（也叫公钥加密），另外一种是对称加密（也叫密钥加密）。所谓非对称加密就是指加密和解密使用的密钥不一样，如下图：

[](http://photo.blog.sina.com.cn/showpic.html#blogid=591c7a8f0102vdoa&url=http://album.sina.com.cn/pic/001Db1PVzy6N1nM7or092)

  HTTPS使用非对称加解密主要有两个作用，一个是密钥协商，另外可以用来做数字签名。所谓密钥协商简单说就是根据双方各自的信息计算得出双方传输内容时对称加解密需要使用的密钥。

       公钥加密过程一般都是服务器掌握私钥，客户端掌握公钥，私钥用来解密，公钥用来加密。公钥可以发放给任何人知道，但是私钥只有服务器掌握，所以公钥加解密非常安全。当然这个安全性必须建立在公钥长度足够大的基础上，目前公钥最低安全长度也需要达到2048位。大的CA也不再支持2048位以下的企业级证书申请。因为1024位及以下的公钥长度已经不再安全，可以被高性能计算机比如量子计算机强行破解。计算性能基本会随着公钥的长度而呈2的指数级下降。

    既然如此为什么还需要对称加密？为什么不一直使用非对称加密算法来完成全部的加解密过程？主要是两点：

    非对称加解密对性能的消耗非常大，一次完全TLS握手，密钥交换时的非对称解密计算量占整个握手过程的95%。而对称加密的计算量只相当于非对称加密的0.1%，如果应用层数据也使用非对称加解密，性能开销太大，无法承受。

    非对称加密算法对加密内容的长度有限制，不能超过公钥长度。比如现在常用的公钥长度是2048位，意味着待加密内容不能超过256个字节。

    目前常用的非对称加密算法是RSA，想强调一点的就是RSA是整个PKI体系及加解密领域里最重要的算法。如果想深入理解HTTPS的各个方面，RSA是必需要掌握的知识点。它的原理主要依赖于三点：

    乘法的不可逆特性。即我们很容易由两个乘数求出它们的积，但是给定一个乘积，很难求出它是由哪两个乘数因子相乘得出的。

    欧拉函数。欧拉函数.varphi(n)是小于或等于n的正整数中与n互质的数的数目

    费马小定理。假如a是一个整数，p是一个质数，那么a^p - a 是p的倍数。

　　这篇中文博客对RSA的原理解释得比较清楚易懂：RSA算法原理。

    RSA算法是第一个也是目前唯一一个既能用于密钥交换又能用于数字签名的算法。另外一个非常重要的密钥协商算法是diffie-hellman(DH).DH不需要预先知道通信双方的信息就能完成密钥的协商，它使用一个素数P的整数乘法群以及原根G，理论依据就是离散对数。

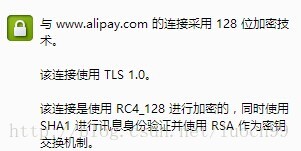
    openssl目前只支持如下密钥交换算法：RSA，DH，ECDH, DHE，ECDHE。各个算法的性能和对速度的影响可以参考后面章节，由于篇幅有限，具体实现不再做详细介绍。

2.1.2 对称加密

    对称加密就是加密和解密都使用的是同一个密钥。如下图：

[](http://photo.blog.sina.com.cn/showpic.html#blogid=591c7a8f0102vdoa&url=http://album.sina.com.cn/pic/001Db1PVzy6N1nV32Lkd6)

    采用非对称密码算法的密钥协商过程结束之后就已经得出了本次会话需要使用的对称密钥。对称加密又分为两种模式：流式加密和分组加密。流式加密现在常用的就是RC4，不过RC4已经不再安全，微软也建议网站尽量不要使用RC4流式加密。支付宝可能没有意识到这一点，也可能是由于其他原因，他们仍然在使用RC4算法和TLS1.0协议。

[](http://photo.blog.sina.com.cn/showpic.html#blogid=591c7a8f0102vdoa&url=http://album.sina.com.cn/pic/001Db1PVzy6N1nYUvIV3d)

     一种新的替代RC4的流式加密算法叫ChaCha20，它是google推出的速度更快，更安全的加密算法。目前已经被android和chrome采用，也编译进了google的开源openssl分支---boring ssl，并且nginx 1.7.4也支持编译boringssl。我目前还没有比较这种算法的性能，但部分资料显示这个算法对性能的消耗比较小，特别是移动端提升比较明显。

       分组加密以前常用的模式是AES-CBC，但是CBC已经被证明容易遭受BEAST和LUCKY13攻击。目前建议使用的分组加密模式是AES-GCM，不过它的缺点是计算量大，性能和电量消耗都比较高，不适用于移动电话和平板电脑。尽管如此，它仍然是我们的优先选择。

2.2 数据完整性

这部分内容相对比较简单，openssl现在使用的完整性校验算法有两种：MD5或者SHA。由于MD5在实际应用中存在冲突的可能性比较大，所以尽量别采用MD5来验证内容一致性。SHA也不能使用SHA0和SHA１，中国山东大学的王小云教授在2005年就牛逼地宣布破解了SHA-1完整版算法。建议使用SHA２算法，即输出的摘要长度超过224位。

2.3 身份验证和授权

       这里主要介绍的就是PKI和数字证书。数字证书有两个作用：

    身份验证。确保客户端访问的网站是经过CA验证的可信任的网站。

    分发公钥。每个数字证书都包含了注册者生成的公钥。在SSL握手时会通过certificate消息传输给客户端。

       这里简单介绍一下数字证书是如何验证网站身份的。

       证书申请者首先会生成一对密钥，包含公钥和密钥，然后把公钥及域名还有CU等资料制作成CSR格式的请求发送给RA，RA验证完这些内容之后（RA会请独立的第三方机构和律师团队确认申请者的身份）再将CSR发送给CA，CA然后制作X.509格式的证书。

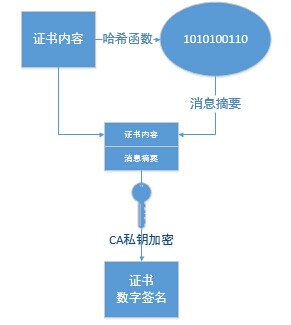
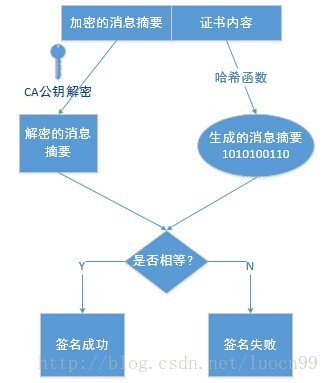
       那好，申请者拿到CA的证书并部署在网站服务器端，那浏览器访问时接收到证书后，如何确认这个证书就是CA签发的呢？怎样避免第三方伪造这个证书？

       答案就是数字签名（digital signature）。数字签名可以认为是一个证书的防伪标签，目前使用最广泛的SHA-RSA数字签名的制作和验证过程如下：

数字签名的签发。首先是使用哈希函数对证书数据哈希，生成消息摘要，然后使用CA自己的私钥对证书内容和消息摘要进行加密。

数字签名的校验。使用CA的公钥解密签名，然后使用相同的签名函数对证书内容进行签名并和服务端的数字签名里的签名内容进行比较，如果相同就认为校验成功。

    图形表示如下：

[](http://photo.blog.sina.com.cn/showpic.html#blogid=591c7a8f0102vdoa&url=http://album.sina.com.cn/pic/001Db1PVzy6N1o4ienr55)  [](http://photo.blog.sina.com.cn/showpic.html#blogid=591c7a8f0102vdoa&url=http://album.sina.com.cn/pic/001Db1PVzy6N1o4g9FDb9)

                           数字签名签发                                                                                 数字签名校验

    这里有几点需要说明：

数字签名签发和校验使用的密钥对是CA自己的公私密钥，跟证书申请者提交的公钥没有关系。

数字签名的签发过程跟公钥加密的过程刚好相反，即是用私钥加密，公钥解密。

现在大的CA都会有证书链，证书链的好处一是安全，保持根CA的私钥离线使用。第二个好处是方便部署和撤销，即如何证书出现问题，只需要撤销相应级别的证书，根证书依然安全。

根CA证书都是自签名，即用自己的公钥和私钥完成了签名的制作和验证。而证书链上的证书签名都是使用上一级证书的密钥对完成签名和验证的。

怎样获取根CA和多级CA的密钥对？它们是否可信？当然可信，因为这些厂商跟浏览器和操作系统都有合作，它们的公钥都默认装到了浏览器或者操作系统环境里。比如firefox就自己维护了一个可信任的CA列表，而chrome和IE使用的是操作系统的CA列表。

       数字证书的费用其实也不高，对于中小网站可以使用便宜甚至免费的数字证书服务（可能存在安全隐患），像著名的verisign公司的证书一般也就几千到几万块一年不等。当然如果公司对证书的需求比较大，定制性要求高，可以建立自己的CA站点，比如google，能够随意签发google相关证书。

**3，HTTPS对速度和性能的影响**

       既然HTTPS非常安全，数字证书费用也不高，那为什么互联网公司不全部使用HTTPS呢？原因主要有两点：

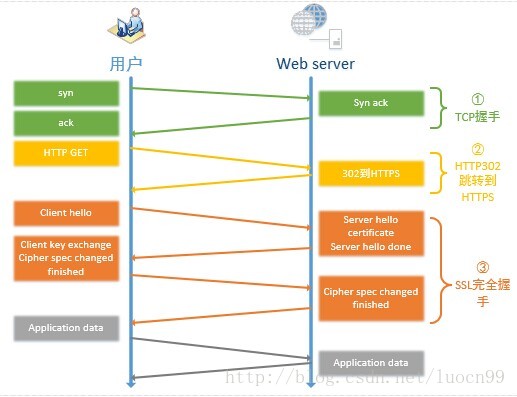
HTTPS对速度的影响非常明显。每个HTTPS连接一般会增加1-3个RTT，加上加解密对性能的消耗，延时还有可能再增加几十毫秒。

HTTPS对CPU计算能力的消耗很严重，完全握手时，web server的处理能力会降低至HTTP的10%甚至以下。

      下面简单分析一下这两点。

3.1 HTTPS对访问速度的影响

       我用一张图来表示一个用户访问使用HTTPS网站可能增加的延时：

[](http://photo.blog.sina.com.cn/showpic.html#blogid=591c7a8f0102vdoa&url=http://album.sina.com.cn/pic/001Db1PVzy6N1o7J1Ld3b)

       HTTPS增加的延时主要体现在三个阶段，包含了上图所示的2和3阶段。

302跳转。为什么需要302？因为用户懒。我想绝大部分网民平时访问百度时都是输入www.baidu.com或者baidu.com吧？很少有输入http://www.baidu.com访问百度搜索的吧？至于直接输入https://www.baidu.com来访问百度的HTTPS服务的就更加少了。所以为了强制用户使用HTTPS服务，只有将用户发起的HTTP请求www.baidu.com302成https://www.baidu.com。这无疑是增加一个RTT的跳转延时。

上图第三阶段的SSL完全握手对延时的影响就更加明显了，这个影响不仅体现在网络传输的RTT上，还包含了数字签名的校验，由于客户端特别是移动端的计算性能弱，增加几十毫秒的计算延时是很常见的。

还有一个延时没有画出来，就是证书的状态检查，现在稍微新一点的浏览器都使用ocsp来检查证书的撤销状态，在拿到服务器的证书内容之后会访问ocsp站点获取证书的状态，检查证书是否撤销。如果这个ocsp站点在国外或者ocsp服务器出现故障，显然会影响这个正常用户的访问速度。不过还好ocsp的检查周期一般都是7天一次，所以这个对速度的影响还不是很频繁。 另外chrome默认是关闭了ocsp及crl功能，最新版的firefox开启了这个功能，如果ocsp返回不正确，用户无法打开访问网站。

      实际测试发现，在没有任何优化的情况下，HTTPS会增加200ms以上的延时。

       那是不是对于这些延时我们就无法优化了呢？显然不是，部分优化方式参考如下：

服务器端配置HSTS，减少302跳转，其实HSTS的最大作用是防止302 HTTP劫持。HSTS的缺点是浏览器支持率不高，另外配置HSTS后HTTPS很难实时降级成HTTP。

设置ssl session 的共享内存cache. 以nginx为例，它目前只支持session cache的单机多进程共享。配置如下：

ssl\_session\_cache    shared:SSL:10m;

如果是前端接入是多服务器架构，这样的session cache是没有作用的，所以需要实现session cache的多机共享机制。我们已经在nginx 1.6.0版本上实现了多机共享的session cache。多机session cache的问题必须要同步访问外部session cache，比如redis。由于openssl目前提供的API是同步的，所以我们正在改进openssl和nginx的异步实现。

配置相同的session ticket key，部署在多个服务器上，这样多个不同的服务器也能产生相同的 session ticket。session ticket的缺点是支持率不广，大概只有40%。而session id是client hello的标准内容，从SSL2.0开始就被全部客户支持。

ssl\_session\_tickets    on;

ssl\_session\_ticket\_key ticket\_keys;

设置ocsp stapling file，这样ocsp的请求就不会发往ca提供的ocsp站点，而是发往网站的webserver。ocsp的配置和生成命令如下：

       ssl\_stapling on;

       ssl\_stapling\_file domain.staple;

上面是nginx配置，如下是ocsp\_stapling\_file的生成命令：

openssl s\_client -showcerts -connect yourdomain:443 < /dev/null | awk -v c=-1 '/-----BEGIN CERTIFICATE-----/{inc=1;c++} inc {print > ("level" c ".crt")} /---END CERTIFICATE-----/{inc=0}'

for i in level?.crt;

do

       openssl x509 -noout -serial -subject -issuer -in "$i";

echo;

done

openssl ocsp -text -no\_nonce -issuer level1.crt -CAfile CAbundle.crt -cert level0.crt -VAfile level1.crt -url $ocsp\_url -respout domain.staple ,其中$ocsp\_url等于ocsp站点的URL，可以通过如下命令求出：for i in level?.crt; do echo "$i:"; openssl x509 -noout -text -in "$i" | grep OCSP; done，如果是证书链，一般是最底层的值。

优先使用ecdhe密钥交换算法，因为它支持PFS(perfect forward secrecy)，能够实现false start。

设置tls record size，最好是能动态调整record size，即连接刚建立时record size设置成msg，连接稳定之后可以将record size动态增加。

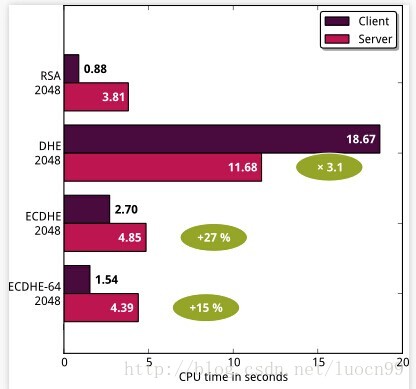
如果有条件的话可以启用tcp fast open。虽然现在没有什么客户端支持。

启用SPDY。SPDY是强制使用HTTPS的，协议比较复杂，需要单独的文章来分析。可以肯定的一点是使用SPDY的请求不仅明显提升了HTTPS速度，甚至比HTTP还要快。在无线WIFI环境下，SPDY比HTTP要快50ms左右，3G环境下比HTTP要快250ms。

3.2 HTTPS 对性能的影响

       HTTPS为什么会严重降低性能？主要是握手阶段时的大数运算。其中最消耗性能的又是密钥交换时的私钥解密阶段（函数是rsa\_private\_decryption）。这个阶段的性能消耗占整个SSL握手性能消耗的95%。

       前面提及了openssl密钥交换使用的算法只有四种：rsa, dhe, ecdhe，dh。dh由于安全问题目前使用得非常少，所以这里可以比较下前面三种密钥交换算法的性能，具体的数据如下：

[](http://photo.blog.sina.com.cn/showpic.html#blogid=591c7a8f0102vdoa&url=http://album.sina.com.cn/pic/001Db1PVzy6N1obR2MOa2)

       上图数据是指完成1000次握手需要的时间，显然时间数值越大表示性能越低。图片和测试方法都可以参考原文，地址如下：http://vincent.bernat.im/en/blog/2011-ssl-perfect-forward-secrecy.html。

       密钥交换步骤是SSL完全握手过程中无法绕过的一个阶段。我们只能采取如下措施：

通过session cache和session ticket提升session reuse率，减少完全握手(full handshake)次数，提升简化握手(abbreviated handshake)率。

出于前向加密和false start的考虑，我们优先配置ecdhe用于密钥交换，但是性能不足的情况下可以将rsa配置成密钥交换算法，提升性能。

       openssl 自带的工具可以计算出对称加密、数字签名及HASH函数的各个性能，所以详细数据我就不再列举，读者可以自行测试 。

结论就是对称加密RC4的性能最快，但是RC4本身不安全，所以还是正常情况下还是采用AES。HASH函数MD5和SHA1差不多。数字签名是ecdsa算法最快，但是支持率不高。

      事实上由于密钥交换在整个握手过程中消耗性能占了95%，而对称加解密的性能消耗不到0.1%，所以server端对称加密的优化收益不大。相反，由于客户端特别是移动端的CPU计算能力本来就比较弱，所以对称加密和数字签名的优化主要是针对移动客户端。

       poly1350是google推出的号称优于aes-gcm的对称加密算法，适用于移动端，可以试用一下。

       最后经过测试，综合安全和性能的最优cipher suite配置是:  ECDHE-RSA-AES128-GCM-SHA256.

      如果性能出现大幅度下降，可以修改配置，提升性能但是弱化了安全性，配置是：rc4-md5，根据openssl的规则，密钥交换和数字签名默认都是使用rsa。

**4，HTTPS的支持率分析**

      分析了百度服务器端一百万的无线访问日志（主要为手机和平板电脑的浏览器），得出协议和握手时间的关系如下：

tls协议版本客户端使用率握手时间 ms

tls 1.224.8)9.496

tls 1.10.9'9.383

tls 1.07407.077

ssl 3.00.3H4.564

      从上表可以发现，ssl3.0速度最慢，不过支持率非常低。tls 1.0支持率最广泛。

      加密套件和握手时间的关系如下：

加密套件客户端使用率握手时间

ECDHE-RSA-AES128-SHA58.5)4.36

ECDHE-RSA-AES128-SHA25621.103.065

DHE-RSA-AES128-SHA16.751.063

ECDHE-RSA-AES128-GCM-SHA2563.7'4.83

显然DHE对速度的影响比较大，ECDHE的性能确实要好出很多，而AES128-GCM对速度也有一点提升。

通过tcpdump分析client hello请求，发现有56.53%的请求发送了session id。也就意味着这些请求都能通过session cache得到复用。其他的一些扩展属性支持率如下：

tls扩展名支持率

server\_name76.99%

session\_tickets38.6%

next\_protocol\_negotiation40.54%

elliptic\_curves 90.6%

ec\_point\_formats90.6%

这几个扩展都非常有意义，解释如下：

server\_name,，即 sni （server name indicator)，有77%的请求会在client hello里面携带想要访问的域名，允许服务端使用一个IP支持多个域名。

next\_protocol\_negotiation，即NPN，意味着有40.54%的客户端支持spdy.

session\_tickets只有38.6%的支持率，比较低。这也是我们为什么会修改nginx主干代码实现session cache多机共享机制的原因。

elliptic\_curves即是之前介绍的ECC（椭圆曲线系列算法），能够使用更小KEY长度实现DH同样级别的安全，极大提升运算性能。

5，结论

       现在互联网上HTTPS的中文资料相对较少，同时由于HTTPS涉及到大量协议、密码学及PKI体系的知识，学习门槛相对较高。另外在具体的实践过程中还有很多坑和待持续改进的地方。希望本文对大家有一些帮助，同时由于我本人在很多地方掌握得也比较粗浅，一知半解，希望大家能多提意见，共同进步。

       最后，为了防止流量劫持，保护用户隐私，大家都使用HTTPS吧，全网站支持。事实上，HTTPS并没有那么难用和可怕，只是你没有好好优化。