[Skip to main content](https://lms.alnafi.com/xblock/block-v1:alnafi+DCCS102+2025_DCCS+type@vertical+block@d5eeb58913544d0392d255ede0b42d09?exam_access=&recheck_access=1&show_bookmark=0&show_title=0&view=student_view#main)

**SSL/TLS Deployment Best Practices**

|  |
| --- |
| SSL/TLS is a deceptively simple technology. It is  easy to deploy, and it just works . . . except that it does not, really. The first part is true—SSL is easy to deploy—but it turns out that it  is not easy to deploy correctly. To ensure that SSL provides the  necessary security, users must put more effort into properly configuring their servers.    In 2009, we began our work on SSL Labs because we  wanted to understand how SSL was used and to remedy the lack of  easy-to-use SSL tools and documentation. We have achieved some of our  goals through our global surveys of SSL usage, as well as the online  assessment tool, but the lack of documentation is still evident. This  document is a first step toward addressing that problem.    Our aim here is to provide clear and concise  instructions to help overworked administrators and programmers spend the minimum time possible to obtain a secure site or web application. In  pursue of clarity, we sacrifice completeness, foregoing certain advanced topics. The focus is on advice that is practical and easy to  understand. For those interested in advanced topics, we provide  references at the end of the guide.    Complete Guide:   * [SSL/TLS Deployment Best Practices](https://github.com/ssllabs/research/wiki/SSL-and-TLS-Deployment-Best-Practices)   **SSL and TLS Deployment Best Practices**            naumanshah03 edited this page on Jan 15, 2020        ·        [14 revisions](https://github.com/ssllabs/research/wiki/SSL-and-TLS-Deployment-Best-Practices/_history)    **Version 1.6-draft (15 January 2020)**  SSL/TLS is a deceptively simple technology. It is easy to deploy, and it just works--except when it does not. The main problem is that encryption is not often easy to deploy *correctly*. To ensure that TLS provides the necessary security, system administrators and developers must put extra effort into properly configuring their servers and developing their applications.  In 2009, we began our work on [SSL Labs](https://www.ssllabs.com/) because we wanted to understand how TLS was used and to remedy the lack of easy-to-use TLS tools and documentation. We have achieved some of our goals through our global surveys of TLS usage, as well as the online assessment tool, but the lack of documentation is still evident. This document is a step toward addressing that problem.  Our aim here is to provide clear and concise instructions to help overworked administrators and programmers spend the minimum time possible to deploy a secure site or web application. In pursuit of clarity, we sacrifice completeness, foregoing certain advanced topics. The focus is on advice that is practical and easy to follow. For those who want more information, Section 6 gives useful pointers.  **1 Private Key and Certificate**  In TLS, all security starts with the server's cryptographic identity; a strong private key is needed to prevent attackers from carrying out  impersonation attacks. Equally important is to have a valid and strong  certificate, which grants the private key the right to represent a  particular hostname. Without these two fundamental building blocks,  nothing else can be secure.  **1.1 Use 2048-Bit Private Keys**  For most web sites, security provided by 2,048-bit RSA keys is  sufficient. The RSA public key algorithm is widely supported, which  makes keys of this type a safe default choice. At 2,048 bits, such keys  provide about 112 bits of security. If you want more security than this, note that RSA keys don't scale very well. To get 128 bits of security,  you need 3,072-bit RSA keys, which are noticeably slower. ECDSA keys  provide an alternative that offers better security and better  performance. At 256 bits, ECDSA keys provide 128 bits of security. A  small number of older clients don't support ECDSA, but modern clients  do. It's possible to get the best of both worlds and deploy with RSA and ECDSA keys simultaneously if you don't mind the overhead of managing  such a setup.  **1.2 Protect Private Keys**  Treat your private keys as an important asset, restricting access to  the smallest possible group of employees while still keeping your  arrangements practical. Recommended policies include the following:   * Generate private keys on a trusted computer with sufficient entropy.  Some CAs offer to generate private keys for you; run away from them. * Password-protect keys from the start to prevent compromise when they  are stored in backup systems. Private key passwords don’t help much in  production because a knowledgeable attacker can always retrieve the keys from process memory. There are hardware devices (called Hardware  Security Modules, or HSMs) that can protect private keys even in the  case of server compromise, but they are expensive and thus justifiable  only for organizations with strict security requirements. * After compromise, revoke old certificates and generate new keys. * Renew certificates yearly, and more often if you can automate the  process. Most sites should assume that a compromised certificate will be impossible to revoke reliably; certificates with shorter lifespans are  therefore more secure in practice. * Unless keeping the same keys is important for public key pinning, you should also generate new private keys whenever you're getting a new  certificate.   **1.3 Ensure Sufficient Hostname Coverage**  Ensure that your certificates cover all the names you wish to use  with a site. Your goal is to avoid invalid certificate warnings, which  confuse users and weaken their confidence.  Even when you expect to use only one domain name, remember that you  cannot control how your users arrive at the site or how others link to  it. In most cases, you should ensure that the certificate works with and without the *www* prefix (e.g., that it works for both *example.com* and [*www.example.com*](http://www.example.com/)). The rule of thumb is that a secure web server should have a certificate that is valid for every DNS name configured to point to it.  Wildcard certificates have their uses, but avoid using them if it  means exposing the underlying keys to a much larger group of people, and especially if doing so crosses team or department boundaries. In other  words, the fewer people there are with access to the private keys, the  better. Also be aware that certificate sharing creates a bond that can  be abused to transfer vulnerabilities from one web site or server to all other sites and servers that use the same certificate (even when the  underlying private keys are different).  Make sure you add all the necessary domain names to Subject  Alternative Name (SAN) since all the latest browsers do not check for  Common Name for validation  **1.4 Obtain Certificates from a Reliable CA**  Select a Certification Authority (CA) that is reliable and serious  about its certificate business and security. Consider the following  criteria when selecting your CA:  **Security posture** All CAs undergo regular audits, but some are more serious about security than others. Figuring out which  ones are better in this respect is not easy, but one option is to  examine their security history, and, more important, how they have  reacted to compromises and if they have learned from their mistakes.  **Business focus** CAs whose activities constitute a  substantial part of their business have everything to lose if something  goes terribly wrong, and they probably won’t neglect their certificate  division by chasing potentially more lucrative opportunities elsewhere.  **Services offered** At a minimum, your selected CA  should provide support for both Certificate Revocation List (CRL) and  Online Certificate Status Protocol (OCSP) revocation methods, with  rock-solid network availability and performance. Many sites are happy  with domain-validated certificates, but you also should consider if  you'll ever require Extended Validation (EV) certificates. In either  case, you should have a choice of public key algorithm. Most web sites  use RSA today, but ECDSA may become important in the future because of  its performance advantages.  **Certificate management options** If you need a large  number of certificates and operate in a complex environment, choose a CA that will give you good tools to manage them.  **Support** Choose a CA that will give you good support if and when you need it.  **Note**  For best results, acquire your certificates well in advance and at  least one week before deploying them to production. This practice (1)  helps avoid certificate warnings for some users who don't have the  correct time on their computers and (2) helps avoid failed revocation  checks with CAs who need extra time to propagate new certificates as  valid to their OCSP responders. Over time, try to extend this "warm-up"  period to 1-3 months. Similarly, don't wait until your certificates are  about to expire to replace them. Leaving an extra several months there  would similarly help with people whose clocks are incorrect in the other direction.  **1.5 Use Strong Certificate Signature Algorithms**  Certificate security depends *(1)* on the strength of the private key that was used to sign the certificate and *(2)* the strength of the hashing function used in the signature. Until  recently, most certificates relied on the SHA1 hashing function, which  is now considered insecure. As a result, we're currently in transition  to SHA256. As of January 2016, you shouldn't be able to get a SHA1  certificate from a public CA. Leaf and intermediate certificates having  SHA1 hashing signature are now considered insecure by browser.  **1.6 Use DNS CAA**  DNS CAA[8] is a standard that allows domain name owners to restrict  which CAs can issue certificates for their domains. In September 2017,  CA/Browser Forum mandated CAA support as part of its certificate  issuance standard baseline requirements. With CAA in place, the attack surface for fraudulent certificates is  reduced, effectively making sites more secure. If the CAs have automated process in place for issuance of certificates, then it should check for DNS CAA record as this would reduce the  improper issuance of certificates.  It is recommended to whitelist a CA by adding a CAA record for your  certificate. Add CA's which you trust for issuing you a certificate.  **2 Configuration**  With correct TLS server configuration, you ensure that your  credentials are properly presented to the site’s visitors, that only  secure cryptographic primitives are used, and that all known weaknesses  are mitigated.  **2.1 Use Complete Certificate Chains**  In most deployments, the server certificate alone is insufficient;  two or more certificates are needed to build a complete chain of trust. A common configuration problem occurs when deploying a server with a  valid certificate, but without all the necessary intermediate  certificates. To avoid this situation, simply use all the certificates  provided to you by your CA in the same sequence.  An invalid certificate chain effectively renders the server  certificate invalid and results in browser warnings. In practice, this  problem is sometimes difficult to diagnose because some browsers can  reconstruct incomplete chains and some can’t. All browsers tend to cache and reuse intermediate certificates.  **2.2 Use Secure Protocols**  There are six protocols in the SSL/TLS family: SSL v2, SSL v3, TLS v1.0, TLS v1.1, TLS v1.2, and TLS v1.3:   * SSL v2 is insecure and must not be used. This protocol version is so  bad that it can be used to attack RSA keys and sites with the same name  even if they are on an entirely different servers (the DROWN attack). * SSL v3 is insecure when used with HTTP (the SSLv3 POODLE attack) and  weak when used with other protocols. It’s also obsolete and shouldn’t be used. * TLS v1.0 and TLS v1.1 are legacy protocol that shouldn't be used, but it's typically still necessary in practice. Its major weakness (BEAST)  has been mitigated in modern browsers, but other problems remain. TLS  v1.0 has been deprecated by PCI DSS. Similarly, TLS v1.0 and TLS v1.1  has been deprecated in January 2020 by modern browsers. Check the SSL  Labs blog [link](https://blog.qualys.com/ssllabs/2018/11/19/grade-change-for-tls-1-0-and-tls-1-1-protocols) * TLS v1.2 and v1.3 are both without known security issues.   TLS v1.2 or TLS v1.3 should be your main protocol because these  version offers modern authenticated encryption (also known as AEAD). If  you don't support TLS v1.2 or TLS v1.3 today, your security is lacking.  In order to support older clients, you may need to continue to  support TLS v1.0 and TLS v1.1 for now. However, you should plan to  retire TLS v1.0 and TLS v1.1 in the near future. For example, the PCI  DSS standard will require all sites that accept credit card payments to  remove support for TLS v1.0 by June 2018. Similarly, modern browsers  will remove the support for TLS v1.0 and TLS v1.1 by January 2020.  Benefits of using TLS v1.3:   * Improved performance i.e improved latency * Improved security * Removed obsolete/insecure features like cipher suites, compression etc.   **2.3 Use Secure Cipher Suites**  To communicate securely, you must first ascertain that you are  communicating directly with the desired party (and not through someone  else who will eavesdrop) and exchanging data securely. In SSL and TLS,  cipher suites define how secure communication takes place. They are  composed from varying building blocks with the idea of achieving  security through diversity. If one of the building blocks is found to be weak or insecure, you should be able to switch to another.  You should rely chiefly on the AEAD suites that provide strong  authentication and key exchange, forward secrecy, and encryption of at  least 128 bits. Some other, weaker suites may still be supported,  provided they are negotiated only with older clients that don't support  anything better.  There are several obsolete cryptographic primitives that *must* be avoided:   * Anonymous Diffie-Hellman (ADH) suites do not provide authentication. * NULL cipher suites provide no encryption. * Export cipher suites are insecure when negotiated in a connection,  but they can also be used against a server that prefers stronger suites  (the FREAK attack). * Suites with weak ciphers (112 bits or less) use encryption that can easily be broken are insecure. * RC4 is insecure. * 64-bit block cipher (3DES / DES / RC2 / IDEA) are weak. * Cipher suites with RSA key exchange are weak i.e. TLS\_RSA   There are several cipher suites that *must* be preferred:   * AEAD (Authenticated Encryption with Associated Data) cipher suites – CHACHA20\_POLY1305, GCM and CCM * PFS (Perfect Forward Secrecy) ciphers – ECDHE\_RSA, ECDHE\_ECDSA, DHE\_RSA, DHE\_DSS, CECPQ1 and all TLS 1.3 ciphers   Use the following suite configuration, designed for both RSA and ECDSA keys, as your starting point:  TLS\_ECDHE\_ECDSA\_WITH\_AES\_128\_GCM\_SHA256  TLS\_ECDHE\_ECDSA\_WITH\_AES\_256\_GCM\_SHA384  TLS\_ECDHE\_ECDSA\_WITH\_AES\_128\_CBC\_SHA  TLS\_ECDHE\_ECDSA\_WITH\_AES\_256\_CBC\_SHA  TLS\_ECDHE\_ECDSA\_WITH\_AES\_128\_CBC\_SHA256  TLS\_ECDHE\_ECDSA\_WITH\_AES\_256\_CBC\_SHA384  TLS\_ECDHE\_RSA\_WITH\_AES\_128\_GCM\_SHA256  TLS\_ECDHE\_RSA\_WITH\_AES\_256\_GCM\_SHA384  TLS\_ECDHE\_RSA\_WITH\_AES\_128\_CBC\_SHA  TLS\_ECDHE\_RSA\_WITH\_AES\_256\_CBC\_SHA  TLS\_ECDHE\_RSA\_WITH\_AES\_128\_CBC\_SHA256  TLS\_ECDHE\_RSA\_WITH\_AES\_256\_CBC\_SHA384  TLS\_DHE\_RSA\_WITH\_AES\_128\_GCM\_SHA256  TLS\_DHE\_RSA\_WITH\_AES\_256\_GCM\_SHA384  TLS\_DHE\_RSA\_WITH\_AES\_128\_CBC\_SHA  TLS\_DHE\_RSA\_WITH\_AES\_256\_CBC\_SHA  TLS\_DHE\_RSA\_WITH\_AES\_128\_CBC\_SHA256  TLS\_DHE\_RSA\_WITH\_AES\_256\_CBC\_SHA256  **Warning**  We recommend that you always first test your TLS configuration in a  staging environment, transferring the changes to the production  environment only when certain that everything works as expected. Please  note that the above is a generic list and that not all systems  (especially the older ones) support all the suites. That's why it's  important to test first.  The above example configuration uses standard TLS suite names. Some  platforms use nonstandard names; please refer to the documentation for  your platform for more details. For example, the following suite names  would be used with OpenSSL:  ECDHE-ECDSA-AES128-GCM-SHA256  ECDHE-ECDSA-AES256-GCM-SHA384  ECDHE-ECDSA-AES128-SHA  ECDHE-ECDSA-AES256-SHA  ECDHE-ECDSA-AES128-SHA256  ECDHE-ECDSA-AES256-SHA384  ECDHE-RSA-AES128-GCM-SHA256  ECDHE-RSA-AES256-GCM-SHA384  ECDHE-RSA-AES128-SHA  ECDHE-RSA-AES256-SHA  ECDHE-RSA-AES128-SHA256  ECDHE-RSA-AES256-SHA384  DHE-RSA-AES128-GCM-SHA256  DHE-RSA-AES256-GCM-SHA384  DHE-RSA-AES128-SHA  DHE-RSA-AES256-SHA  DHE-RSA-AES128-SHA256  DHE-RSA-AES256-SHA256  **2.4 Select Best Cipher Suites**  In SSL v3 and later protocol versions, clients submit a list of  cipher suites that they support, and servers choose one suite from the  list to use for the connection. Not all servers do this well, however;  some will select the first supported suite from the client's list.  Having servers actively select the best available cipher suite is  critical for achieving the best security.  **2.5 Use Forward Secrecy**  Forward secrecy (sometimes also called perfect forward secrecy) is a  protocol feature that enables secure conversations that are not  dependent on the server’s private key. With cipher suites that do not  provide forward secrecy, someone who can recover a server’s private key  can decrypt *all* earlier recorded encrypted conversations. You  need to support and prefer ECDHE suites in order to enable forward  secrecy with modern web browsers. To support a wider range of clients,  you should also use DHE suites as fallback after ECDHE. Avoid the RSA  key exchange unless absolutely necessary. My proposed default  configuration in Section 2.3 contains only suites that provide forward  secrecy.  **2.6 Use Strong Key Exchange**  For the key exchange, public sites can typically choose between the  classic ephemeral Diffie-Hellman key exchange (DHE) and its elliptic  curve variant, ECDHE. There are other key exchange algorithms, but  they're generally insecure in one way or another. The RSA key exchange  is still very popular, but it doesn't provide forward secrecy.  In 2015, a group of researchers published new attacks against DHE;  their work is known as the Logjam attack.[2] The researchers discovered  that lower-strength DH key exchanges (e.g., 768 bits) can easily be  broken and that some well-known 1,024-bit DH groups can be broken by  state agencies. To be on the safe side, if deploying DHE, configure it  with at least 2,048 bits of security. Some older clients (e.g., Java 6)  might not support this level of strength. For performance reasons, most  servers should prefer ECDHE, which is both stronger and faster. The secp256r1 named curve (also known as P-256) is a good choice in this case.  **2.7 Mitigate Known Problems**  There have been several serious attacks against SSL and TLS in recent years, but they should generally not concern you if you're running  up-to-date software and following the advice in this guide. (If you're  not, I'd advise testing your systems using SSL Labs and taking it from  there.) However, nothing is perfectly secure, which is why it is a good  practice to keep an eye on what happens in security. Promptly apply  vendor patches if and when they become available; otherwise, rely on  workarounds for mitigation.  **3 Performance**  Security is our main focus in this guide, but we must also pay  attention to performance; a secure service that does not satisfy  performance criteria will no doubt be dropped. With proper  configuration, TLS can be quite fast. With modern protocols—for example, HTTP/2—it might even be faster than plaintext communication.  **3.1 Avoid Too Much Security**  The cryptographic handshake, which is used to establish secure  connections, is an operation for which the cost is highly influenced by  private key size. Using a key that is too short is insecure, but using a key that is too long will result in “too much” security and slow  operation. For most web sites, using RSA keys stronger than 2,048 bits  and ECDSA keys stronger than 256 bits is a waste of CPU power and might  impair user experience. Similarly, there is little benefit to increasing the strength of the ephemeral key exchange beyond 2,048 bits for DHE  and 256 bits for ECDHE. There are no clear benefits of using encryption  above 128 bits.  **3.2 Use Session Resumption**  Session resumption is a performance-optimization technique that makes it possible to save the results of costly cryptographic operations and  to reuse them for a period of time. A disabled or nonfunctional session  resumption mechanism may introduce a significant performance penalty.  **3.3 Use WAN Optimization and HTTP/2**  These days, TLS overhead doesn't come from CPU-hungry cryptographic  operations, but from network latency. A TLS handshake, which can start  only after the TCP handshake completes, requires a further exchange of  packets and is more expensive the further away you are from the server.  The best way to minimize latency is to avoid creating new connections—in other words, to keep existing connections open for a long time  (keep-alives). Other techniques that provide good results include  supporting modern protocols such as HTTP/2 and using WAN optimization  (usually via content delivery networks).  **3.4 Cache Public Content**  When communicating over TLS, browsers might assume that all traffic  is sensitive. They will typically use the memory to cache certain  resources, but once you close the browser, all the content may be lost.  To gain a performance boost and enable long-term caching of some  resources, mark public resources (e.g., images) as public.  **3.5 Use OCSP Stapling**  OCSP stapling is an extension of the OCSP protocol that delivers  revocation information as part of the TLS handshake, directly from the  server. As a result, the client does not need to contact OCSP servers  for out-of-band validation and the overall TLS connection time is  significantly reduced. OCSP stapling is an important optimization  technique, but you should be aware that not all web servers provide  solid OCSP stapling implementations. Combined with a CA that has a slow  or unreliable OCSP responder, such web servers might create performance  issues. For best results, simulate failure conditions to see if they  might impact your availability.  **3.6 Use Fast Cryptographic Primitives**  In addition to providing the best security, my recommended cipher  suite configuration also provides the best performance. Whenever  possible, use CPUs that support hardware-accelerated AES. After that, if you really want a further performance edge (probably not needed for  most sites), consider using ECDSA keys.  **4 HTTP and Application Security**  The HTTP protocol and the surrounding platform for web application  delivery continued to evolve rapidly after SSL was born. As a result of  that evolution, the platform now contains features that can be used to  defeat encryption. In this section, we list those features, along with  ways to use them securely.  **4.1 Encrypt Everything**  The fact that encryption is optional is probably one of the biggest security problems today. We see the following problems:   * No TLS on sites that need it * Sites that have TLS but that do not enforce it * Sites that mix TLS and non-TLS content, sometimes even within the same page * Sites with programming errors that subvert TLS   Although many of these problems can be mitigated if you know exactly  what you’re doing, the only way to reliably protect web site  communication is to enforce encryption throughout—without exception.  **4.2 Eliminate Mixed Content**  Mixed-content pages are those that are transmitted over TLS but  include resources (e.g., JavaScript files, images, CSS files) that are  not transmitted over TLS. Such pages are not secure. An active  man-in-the-middle (MITM) attacker can piggyback on a single unprotected  JavaScript resource, for example, and hijack the entire user session.  Even if you follow the advice from the previous section and encrypt your entire web site, you might still end up retrieving some resources  unencrypted from third-party web sites.  **4.3 Understand and Acknowledge Third-Party Trust**  Web sites often use third-party services activated via JavaScript  code downloaded from another server. A good example of such a service is Google Analytics, which is used on large parts of the Web. Such  inclusion of third-party code creates an implicit trust connection that  effectively gives the other party full control over your web site. The  third party may not be malicious, but large providers of such services  are increasingly seen as targets. The reasoning is simple: if a large  provider is compromised, the attacker is automatically given access to  all the sites that depend on the service.  If you follow the advice from Section 4.2, at least your third-party  links will be encrypted and thus safe from MITM attacks. However, you  should go a step further than that: learn what services you use and  remove them, replace them with safer alternatives, or accept the risk of their continued use. A new technology called subresource integrity  (SRI) could be used to reduce the potential exposure via third-party  resources.[3]  **4.4 Secure Cookies**  To be properly secure, a web site requires TLS, but also that all its cookies are explicitly marked as secure when they are created. Failure  to secure the cookies makes it possible for an active MITM attacker to  tease some information out through clever tricks, even on web sites that are 100% encrypted. For best results, consider adding cryptographic  integrity validation or even encryption to your cookies.  **4.5 Secure HTTP Compression**  The 2012 CRIME attack showed that TLS compression can't be  implemented securely. The only solution was to disable TLS compression  altogether. The following year, two further attack variations followed.  TIME and BREACH focused on secrets in HTTP response bodies compressed  using HTTP compression. Unlike TLS compression, HTTP compression is a  necessity and can't be turned off. Thus, to address these attacks,  changes to application code need to be made.[4]  TIME and BREACH attacks are not easy to carry out, but if someone is  motivated enough to use them, the impact is roughly equivalent to a  successful Cross-Site Request Forgery (CSRF) attack.  **4.6 Deploy HTTP Strict Transport Security**  HTTP Strict Transport Security (HSTS) is a safety net for TLS. It was designed to ensure that security remains intact even in the case of  configuration problems and implementation errors. To activate HSTS  protection, you add a new response header to your web sites. After that, browsers that support HSTS (all modern browsers at this time) enforce  it.  The goal of HSTS is simple: after activation, it does not allow any  insecure communication with the web site that uses it. It achieves this  goal by automatically converting all plaintext links to secure ones. As a bonus, it also disables click-through certificate warnings.  (Certificate warnings are an indicator of an active MITM attack. Studies have shown that most users click through these warnings, so it is in  your best interest to never allow them.)  Adding support for HSTS is the single most important improvement you  can make for the TLS security of your web sites. New sites should always be designed with HSTS in mind and the old sites converted to support it wherever possible and as soon as possible. For best security, consider  using HSTS preloading,[5] which embeds your HSTS configuration in modern browsers, making even the first connection to your site secure.  The following configuration example activates HSTS on the main  hostname and all its subdomains for a period of one year, while also  allowing preloading:  Strict-Transport-Security: max-age=31536000; includeSubDomains; preload  **4.7 Deploy Content Security Policy**  Content Security Policy (CSP) is a security mechanism that web sites  can use to restrict browser operation. Although initially designed to  address Cross-Site Scripting (XSS), CSP is constantly evolving and  supports features that are useful for enhancing TLS security. In  particular, it can be used to restrict mixed content when it comes to  third-party web sites, for which HSTS doesn't help.  To deploy CSP to prevent third-party mixed content, use the following configuration:  Content-Security-Policy: default-src https: 'unsafe-inline' 'unsafe-eval';  connect-src https: wss:  **Note**  This is not the best way to deploy CSP. In order to provide an  example that doesn't break anything except mixed content, I had to  disable some of the default security features. Over time, as you learn  more about CSP, you should change your policy to bring them back.  **4.8 Do Not Cache Sensitive Content**  All sensitive content must be communicated only to the intended  parties and treated accordingly by all devices. Although proxies do not  see encrypted traffic and cannot share content among users, the use of  cloud-based application delivery platforms is increasing, which is why  you need to be very careful when specifying what is public and what is  not.  **4.9 Consider Other Threats**  TLS is designed to address only one aspect of  security—confidentiality and integrity of the communication between you  and your users—but there are many other threats that you need to deal  with. In most cases, that means ensuring that your web site does not  have other weaknesses.  **5 Validation**  With many configuration parameters available for tweaking, it is  difficult to know in advance what impact certain changes will have.  Further, changes are sometimes made accidentally; software upgrades can  introduce changes silently. For that reason, we advise that you use a  comprehensive SSL/TLS assessment tool initially to verify your  configuration to ensure that you start out secure, and then periodically to ensure that you stay secure. For public web sites, we recommend the  free SSL Labs server test.[6]  **6 Advanced Topics**  The following advanced topics are currently outside the scope of our  guide. They require a deeper understanding of SSL/TLS and Public Key  Infrastructure (PKI), and they are still being debated by experts.  **6.1 Public Key Pinning**  Public key pinning is designed to give web site operators the means  to restrict which CAs can issue certificates for their web sites. This  feature has been deployed by Google for some time now (hardcoded into  their browser, Chrome) and has proven to be very useful in preventing  attacks and making the public aware of them. In 2014, Firefox also added support for hardcoded pinning. A standard called Public Key Pinning  Extension for HTTP[7] is now available. Public key pinning addresses the biggest weakness of PKI (the fact that any CA can issue a certificate  for any web site), but it comes at a cost; deploying requires  significant effort and expertise, and creates risk of losing control of  your site (if you end up with invalid pinning configuration). You should consider pinning largely only if you're managing a site that might be  realistically attacked via a fraudulent certificate.  **6.2 DNSSEC and DANE**  Domain Name System Security Extensions (DNSSEC) is a set of  technologies that add integrity to the domain name system. Today, an  active network attacker can easily hijack any DNS request and forge  arbitrary responses. With DNSSEC, all responses can be cryptographically tracked back to the DNS root. DNS-based Authentication of Named  Entities (DANE) is a separate standard that builds on top of DNSSEC to  provide bindings between DNS and TLS. DANE could be used to augment the  security of the existing CA-based PKI ecosystem or bypass it altogether.  Even though not everyone agrees that DNSSEC is a good direction for  the Internet, support for it continues to improve. Browsers don't yet  support either DNSSEC or DANE (preferring similar features provided by  HSTS and HPKP instead), but there is some indication that they are  starting to be used to improve the security of email delivery.  **7 Changes**  The first release of this guide was on 24 February 2012. This section tracks the document changes over time, starting with version 1.3.  **Version 1.3 (17 September 2013)**  The following changes were made in this version:   * Recommend replacing 1024-bit certificates straight away. * Recommend against supporting SSL v3. * Remove the recommendation to use RC4 to mitigate the BEAST attack server-side. * Recommend that RC4 is disabled. * Recommend that 3DES is disabled in the near future. * Warn about the CRIME attack variations (TIME and BREACH). * Recommend supporting forward secrecy. * Add discussion of ECDSA certificates.   **Version 1.4 (8 December 2014)**  The following changes were made in this version:   * Discuss SHA1 deprecation and recommend migrating to the SHA2 family. * Recommend that SSL v3 is disabled and mention the POODLE attack. * Expand Section 3.1 to cover the strength of the DHE and ECDHE key exchanges. * Recommend OCSP Stapling as a performance-improvement measure, promoting it to Section 3.5.   **Version 1.5 (8 June 2016)**  The following changes were made in this version:   * Refreshed the entire document to keep up with the times. * Recommended use of authenticated cipher suites. * Spent more time discussing key exchange strength and the Logjam attack. * Removed the recommendation to disable client-initiated renegotiation. Modern software does this anyway, and it might be impossible or  difficult to disable it with something older. At the same time, the DoS  vector isn't particularly strong. Overall, I feel it's better to spend  available resources fixing something else. * Added a warning about flaky OCSP stapling implementations. * Added mention of subresource integrity enforcement. * Added mention of cookie integrity validation and encryption. * Added mention of HSTS preloading. * Recommended using CSP for better handling of third-party mixed content. * Mentioned FREAK, Logjam, and DROWN attacks. * Removed the section that discussed mitigation of various TLS attacks, which are largely obsolete by now, especially if the advice presented  here is followed. Moved discussion of CRIME variants into a new section. * Added a brief discussion of DNSSEC and DANE to the Advanced section.   **Version 1.6 (15 January 2020)**  The following changes were made in this version:   * Refreshed the entire document to keep up with the times. * Added details to use SAN (Subject Alternative Names) since the Common Name is deprecated by latest browsers. * SHA1 signature deprecation for leaf and intermediate certificate * Added DNS CAA information, recommened the use of it. * Added information about the extra download of missing intermediate certificate and the sequence of it. * Recommended the use of TLS 1.3 * Recommended not to use the legacy protocol TLS v1.0 and TLS v1.1 * Improved the secure cipher suites section with more information and newly discovered weak/insecure cipher. * Updated HSTS preload footnotes link.   **Acknowledgments**  Special thanks to Marsh Ray, Nasko Oskov, Adrian F. Dimcev, and Ryan  Hurst for their valuable feedback and help in crafting the initial  version of this document. Also thanks to many others who generously  share their knowledge of security and cryptography with the world. The  guidelines presented here draw on the work of the entire security  community.  **About SSL Labs**  SSL Labs ([www.ssllabs.com](https://github.com/ssllabs/research/wiki/www.ssllabs.com)) is Qualys’s research effort to understand SSL/TLS and PKI as well as to provide tools and documentation to assist with assessment and  configuration. Since 2009, when SSL Labs was launched, hundreds of  thousands of assessments have been performed using the free online  assessment tool. Other projects run by SSL Labs include periodic  Internet-wide surveys of TLS configuration and SSL Pulse, a monthly scan of about 150,000 most popular TLS-enabled web sites in the world.  **About Qualys**  Qualys, Inc. (NASDAQ: QLYS) is a pioneer and leading provider of  cloud-based security and compliance solutions with over 9,300 customers  in more than 100 countries, including a majority of each of the Forbes  Global 100 and Fortune 100. The Qualys Cloud Platform and integrated  suite of solutions help organizations simplify security operations and  lower the cost of compliance by delivering critical security  intelligence on demand and automating the full spectrum of auditing,  compliance and protection for IT systems and web applications. Founded  in 1999, Qualys has established strategic partnerships with leading  managed service providers and consulting organizations including  Accenture, BT, Cognizant Technology Solutions, Deutsche Telekom,  Fujitsu, HCL, HP Enterprise, IBM, Infosys, NTT, Optiv, SecureWorks, Tata Communications, Verizon and Wipro. The company is also a founding  member of the Cloud Security Alliance (CSA). For more information,  please visit [www.qualys.com](http://www.qualys.com/).  [1] [Transport Layer Security (TLS) Parameters](http://www.iana.org/assignments/tls-parameters/tls-parameters.xhtml) (IANA, retrieved 18 March 2016)  [2] [Weak Diffie-Hellman and the Logjam Attack](https://weakdh.org/) (retrieved 16 March 2016)  [3] [Subresource Integrity](https://developer.mozilla.org/en-US/docs/Web/Security/Subresource_Integrity) (Mozilla Developer Network, retrieved 16 March 2016)  [4] [Defending against the BREACH Attack](https://community.qualys.com/blogs/securitylabs/2013/08/07/defending-against-the-breach-attack) (Qualys Security Labs; 7 August 2013)  [5] [HSTS Preload List](https://hstspreload.org/) (Google developers, retrieved 16 March 2016)  [6] [SSL Labs](https://www.ssllabs.com/) (retrieved 16 March 2016)  [7] [RFC 7469: Public Key Pinning Extension for HTTP](https://datatracker.ietf.org/doc/rfc7469/) (Evans et al, April 2015)  [8] [RFC 6844: DNS CAA](https://tools.ietf.org/html/rfc6844) (Evans et al, January 2013) |