Introduction

Transport Layer Security (TLS) and Hypertext Transfer Protocol Secure (HTTPS) are protocols that provide encryption and authentication to reassure people (herein referred to as users) that they are connecting to websites they intend to, and that their interactions are not able to be viewed or modified. These protocols are underpinned by cryptographic documents, known as certificates, which can authenticate the identity of websites.

Public Key Infrastructure arrangements are a key component of the assurance process that enables websites to be authenticated. This is achieved by trusted Certificate Authorities (CAs) that vouch for website identities and attest the public key contained in a certificate belongs to the entity noted on the certificate.

Using the TLS and HTTPS configuration guidelines outlined in this document will help strengthen website encryption and authentication. Further, all public-facing websites and Hypertext Transfer Protocol (HTTP)-enabled application programming interfaces (APIs) should use HTTPS to protect users’ confidentiality.

How certificates, TLS and HTTPS work

Certificates

Many internet protocols use X.509\(^1\) certificates to allow web browsers to authenticate websites using Public Key Cryptography\(^2\). When a user attempts to access a HTTPS-enabled website, the web server sends the user’s web browser its public key contained in a certificate, and demonstrates it has the corresponding private key. The web browser then checks that the certificate was issued by a trusted CA, is valid and was issued to the domain of the website the user is accessing.

To receive a certificate, a website owner will need to apply to a CA and demonstrate that they have control over the domain for which they are requesting a certificate.

Certificate Authorities

Common web browsers (e.g. Google Chrome, Firefox, Microsoft Edge and Safari) maintain their own list of trusted CAs. If a user visits a website which offers a certificate signed by a trusted CA, the web browser will accept the certificate without displaying a trust error. However, if a user visits a website which offers a certificate which is not signed by a trusted CA, the web browser will display an error message and refuse to make an encrypted connection until the user acknowledges the risk.

---

\(^1\) [https://en.wikipedia.org/wiki/X.509](https://en.wikipedia.org/wiki/X.509)

\(^2\) [https://en.wikipedia.org/wiki/Public-key_cryptography](https://en.wikipedia.org/wiki/Public-key_cryptography)
In practice, common web browsers’ trusted CAs are closely aligned. Further, when one web browser developer chooses to remove a trusted CA they will often coordinate with other web browser developers, or other web browser developers will shortly follow suit.

If the CA which has signed a website’s certificate is removed from a web browsers’ trusted CA list, the website’s users will begin to receive security error messages when attempting to connect to the website. This may cause some disruption and embarrassment to the organisation that owns the website until such time that the certificate is replaced.

When choosing a CA to issue a certificate for a website or HTTP-enabled API, organisations should consider the reputation and history of the CA, and other organisations that support or utilise the CA. Finally, the price of certificates is not necessarily an indicator of security, quality or longevity.

**Types of certificates**

CAs typically offer three products: Domain Validation (DV) certificates, Organisation Validation (OV) certificates and Extended Validation (EV) certificates:

- DV certificates can be issued to anyone who demonstrates control over a domain.
- OV certificates include more details about the organisation to whom they are issued.
- EV certificates are considered to be of higher identity assurance than OV and DV certificates as they require an organisation to provide extensive documentation to prove identity and ownership of a domain. Procuring EV certificates takes greater time and resources compared to DV certificates. The issuance of these certificates cannot be automated.

DV certificates provide sufficient confidentiality and authentication to suit most websites and can be obtained for free, or at a price. Additional security or assurances are not achieved by paying for a DV certificate.

OV certificates are priced above DV certificates and allow for more organisational data to be added to the certificate. Users won’t know the difference between a DV and an OV certificate without closely examining certificate details.

EV certificates are the most expensive and require additional identity checks before one can be issued. Common web browsers such as Google Chrome, Firefox and Safari would provide users with additional visual cues when an EV certificate was being used, but this no longer occurs³.

The three certificates types are differentiated by the level of assurance that a domain is owned by an organisation, not by the level of encryption offered. A more expensive certificate does not provide a greater level of confidentiality.

**Encryption algorithms**

Certificate encryption and hashing algorithms should be selected from the Australian Government Information Security Manual (ISM). These can be specified with a CA when requesting a certificate. To change the algorithms being used a new certificate will need to be issued.

**Transport Layer Security**

TLS is a cryptographic protocol that allows for end-to-end encrypted communications over a network. It is used in a variety of applications and builds on the deprecated Secure Socket Layer (SSL) protocol developed by Netscape in 1994.

Websites running SSL or versions of TLS earlier than TLS 1.3 may be susceptible to compromise. If organisations need to use older versions of TLS care should be taken to select cipher suites and configure protocol features in accordance with this guide and the ISM. SSL should not be used.

---

Cipher suites

A key element of understanding how TLS works is understanding what a cipher suite is. A cipher suite is a set of algorithms that help secure a TLS connection. Cipher suites generally include three different algorithms:

- a key establishment algorithm for securely establishing a symmetric key between two devices
- a bulk encryption algorithm (which uses the symmetric key) for encrypting traffic that is sent over the connection
- a Message Authentication Code algorithm for providing assurance that traffic is not modified in transit.

There are many different cipher suites. Selecting the right one is important as weak cipher suites increase the risk to users’ confidentiality.

TLS 1.3 removed vulnerable cipher suites found in TLS 1.2, while introducing stronger cipher suites. Advice on acceptable cipher suites is outlined in Annex A.

TLS handshake process

The following is a simplified explanation of the TLS handshake process:

- the client and server agree on the cryptographic protocol (e.g. TLS 1.3) and cipher suite
- the client authenticates the server:
  - the server offers its certificate and proves that it holds the private key by signing a message which the client can verify using the public key contained in the certificate
  - the client verifies the identity of the server by checking the name of the server matches the name on the certificate
  - the client verifies that the certificate is valid by being in date, not revoked and issued by a CA which the client trusts
- the server and client exchange the symmetric key.

More information on the TLS handshake process is available from Cloudflare.

Perfect Forward Secrecy

Perfect Forward Secrecy (PFS) is a feature of certain cipher suites that reduces the risk posed by compromised session keys. With PFS, individual sessions have unique session keys. As a result, a compromised session key cannot be used to decrypt other sessions. This means that attacks that rely on long-term storage of encrypted data become infeasible. Organisations should only use cipher suites that support PFS.

Secure renegotiation and client-initiated renegotiation

TLS 1.3 does not use renegotiation, however, if using TLS 1.2 or earlier, renegotiation may be required under certain circumstances. For example, when a session has expired but parties wish to send more data, a peer wants to change cipher suites or there is a need for the parties to perform authentication. Unfortunately renegotiation is susceptible to person-in-the-middle attacks. For TLS 1.2 or earlier, secure renegotiation should be enabled to reduce susceptibility to person-in-the-middle attacks.

4 https://www.thesslstore.com/blog/cipher-suites-algorithms-security-settings/
6 https://nvd.nist.gov/vuln/detail/CVE-2009-3555
Client-initiated renegotiation, secure or otherwise, imposes a performance impact on web servers. A malicious client can send many renegotiation requests to consume web server resources causing a Denial of Service\(^7\). For TLS 1.2 or earlier, client-initiated renegotiation should be disabled to prevent Denial of Service attacks. For more information on this subject see McAfee’s *Tips for Securing SSL Renegotiation* advice\(^8\).

**TLS compression**

TLS compression was used to decrease the bandwidth of TLS communications. However, TLS compression has been found to inadvertently leak information, as illustrated by the CRIME exploit\(^9\). TLS compression should be disabled.

**Review TLS settings regularly**

Best practice TLS settings and cipher suites change as new standards are released, computing power increases and security vulnerabilities are discovered. Organisations should review their TLS and cipher suite configurations annually, or whenever major security vulnerabilities are publically disclosed, to ensure that protection remains effective and in line with their customers’ expectations.

**Hypertext Transfer Protocol Secure**

HTTPS allows for secure communication across an untrusted network, reducing the ability of adversaries to monitor a user’s activity. Importantly, lack of HTTPS can erode trust in a website as common web browsers will alert users when visiting such websites\(^10\), and again when they try to enter any information on such websites\(^11\). It should also be noted that some web browsers will block content that is delivered over HTTP when connecting to a website over HTTPS\(^12\).

All website content should be delivered using HTTPS, and any attempted access to resources using HTTP should be automatically redirected to the same resource over HTTPS. In addition, applications which make use of HTTP-enabled APIs should also use TLS to protect the confidentiality and integrity of information in transit.

There are many myths suggesting reasons for not using HTTPS. These are debunked at *Does My Site Need HTTPS*\(^13\).

**HTTP Strict Transport Security**

HTTP Strict Transport Security (HSTS) is a web security policy mechanism that helps protect users. It achieves this by allowing web servers to tell web browsers that they should only interact with a web server over HTTPS. As such, web browsers will dynamically adjust any HTTP requests to HTTPS requests. HSTS also helps to protect against eavesdropping, person-in-the-middle attacks and active network attacks. Organisations should use HSTS to protect users’ confidentiality.

**How to implement certificates, TLS and HTTPS**

**Which cipher suites and encryptions methods to support**

Picking strong cipher suites is important for users’ confidentiality. For the certificate signature algorithm, organisations are encouraged to use Elliptic Curve Digital Signature Algorithm (ECDSA) (256-bit or larger) or Rivest-Shamir-Adleman.
(RSA) (2048-bit or larger). If using elliptic curve cryptography, a curve from Federal Information Processing Standard 186-4 should be used.

Organisation should not use any cipher suite that uses the following algorithms as they have cryptographic weaknesses:

- Rivest Cipher 2
- Rivest Cipher 4
- Message-Digest 5
- Data Encryption Standard
- EXPORT
- NULL
- Secure Hash Algorithm 1
- Anonymous Diffie-Hellman

Advice on acceptable cipher suites is outlined in Annex A.

**Pick and install a certificate**

As mentioned previously, the three certificate types provide the same level of security differing only in the level of assurance that a domain is owned by an organisation. Unless there is a business requirement for a higher assurance certificate, a free DV certificate should be used. In addition, using a CA that provides support for the Automated Certificate Management Environment (ACME) protocol\(^\text{14}\) will enable support for HTTPS in a low maintenance manner.

Let’s Encrypt\(^\text{15}\) is an example of a free DV certificate provider that supports ACME. A case study on setting up Let’s Encrypt with automatic renewal of certificates is provided at the end of this document. For more information on Let’s Encrypt see their Getting Started guide\(^\text{16}\).

**Automate renewal of certificate**

Organisations should endeavour to automate certificate renewal. This can be done using a CA that supports the ACME protocol. Automation of certificate renewal minimises the risk that a website become insecure, or even inaccessible, if its certificate isn’t renewed in time\(^\text{17}\).

Certbot\(^\text{18}\) is a common ACME client that assists in obtaining and installing Let’s Encrypt certificates. It is simple, has comprehensive documentation and works on many platforms. Certbot can setup HTTP redirects, HSTS and load all resources through HTTPS. Alternatively, a list of alternate ACME clients is available on the Let’s Encrypt website\(^\text{19}\).

**Manual setup**

If choosing to install a certificate manually, a Certificate Signing Request (CSR) will need to be created, the corresponding certificate will need to be ordered and then the certificate will need to be installed.


\(^{15}\) [https://letsencrypt.org/](https://letsencrypt.org/)

\(^{16}\) [https://letsencrypt.org/getting-started/](https://letsencrypt.org/getting-started/)


\(^{18}\) [https://certbot.eff.org/](https://certbot.eff.org/)

\(^{19}\) [https://letsencrypt.org/docs/client-options/](https://letsencrypt.org/docs/client-options/)
There are several resources available to assist with this process such as wikiHow’s text tutorial20 and GlobalSign’s video tutorials:

- creating a CSR using Microsoft Management Console21
- creating a CSR in Microsoft IIS 1022
- creating a Java Key Store and generating a CSR23
- creating a CSR in Apache OpenSSL24
- installing a TLS certificate on Microsoft IIS25
- installing a TLS certificate on an Apache Tomcat server26
- installing a TLS certificate on an NGINX server27.

The SSL Store also has advice on creating a CSR28 and installing a TLS certificate29.

Once manual setup has been completed, a TLS tester such as SSL Labs30 can check the implementation.

Redirect HTTP requests

To ensure that users trying to access a website over HTTP are redirected to HTTPS, the web server’s or reverse proxy’s configuration should be edited. While syntax will differ between web server software, an example for Apache HTTP Server is shown below that causes an HTTP 301 redirect whenever the HTTP version of the website is requested.

- RewriteEngine On
- RewriteCond %{HTTPS} off
- RewriteRule ^(.*)$ https://%{HTTP_HOST}%{REQUEST_URI} [L,R=301]

Note, rewrite rules can be complex and it is important to test configurations before and after deployment.

Alternatively, redirection can be achieved in IIS by using the Uniform Resource Locator (URL) Rewrite Module31 while Let’s Encrypt and Certbot can setup redirects automatically when a certificate is obtained.

Regardless of how HTTP to HTTPS redirects are implemented, websites should redirect to the HTTPS version of the website before redirecting elsewhere. For example, if redirecting requests for website.com.au to www.website.com.au, make sure to first redirect to the HTTPS version of website.com.au before redirecting to www.website.com.au. This is required for HSTS to operate correctly. Once the redirection configuration has been tested and confirmed, it is best to use a 301 (permanent) redirect as this typically improves search engine rankings.

21 [https://www.youtube.com/watch?v=30Jnl06PI74](https://www.youtube.com/watch?v=30Jnl06PI74)
22 [https://www.youtube.com/watch?v=mU7iuX4Cy3Y](https://www.youtube.com/watch?v=mU7iuX4Cy3Y)
23 [https://www.youtube.com/watch?v=KPkPWx07zA8](https://www.youtube.com/watch?v=KPkPWx07zA8)
24 [https://www.youtube.com/watch?v=ZAEG9p1_N6_Q](https://www.youtube.com/watch?v=ZAEG9p1_N6_Q)
25 [https://www.youtube.com/watch?v=nB3TFdOkcU](https://www.youtube.com/watch?v=nB3TFdOkcU)
26 [https://www.youtube.com/watch?v=kud2Xsz9Bis](https://www.youtube.com/watch?v=kud2Xsz9Bis)
27 [https://www.youtube.com/watch?v=RwZFDcDuyfg](https://www.youtube.com/watch?v=RwZFDcDuyfg)
28 [https://www.thesslstore.com/knowledgebase/ssl-install](https://www.thesslstore.com/knowledgebase/ssl-install)
29 [https://www.thesslstore.com/knowledgebase/ssl-generate](https://www.thesslstore.com/knowledgebase/ssl-generate)
30 [https://www.ssllabs.com/ssltest/](https://www.ssllabs.com/ssltest/)
Check for mixed content

While users may be able to access a website over HTTPS, parts of the website may still be retrieved using HTTP, such as images or embedded content. Most web browsers will block this mixed content. To check if a website has mixed content issues, either use Firefox’s Web Console to view individual pages\(^{32}\) or use free online services such as Missing Padlock – SSL Checker\(^{33}\).

Improving search engine optimisation

Moving a website to HTTPS will result in search engines treating the website as new, which could place it lower in search engine rankings. However, moving to HTTPS can also improve search engine rankings as many search engines place websites loaded over HTTPS higher in their search results\(^{34}\). To minimise any negative impact on websites’ search engine optimisations, new HTTPS websites can be added and verified, for example, in the Google Search Console\(^{35}\). This will re-crawl the website and submit a new XML sitemap for the HTTPS version\(^{36}\).

HSTS header

Even when HTTPS configurations and HTTP redirects are setup correctly, a HSTS header should still be used. This will ensure that for a specified period of time users will only be able to connect to a website using HTTPS. This is recommended to prevent adversaries from intercepting users’ initial HTTP requests. If using Let’s Encrypt and Certbot, sending HSTS headers can be setup automatically when obtaining a certificate.

Further information on HSTS implementation is available from GlobalSign\(^{37}\).

Sign up for the HSTS Preload List

Signing up for the HSTS Preload List\(^{38}\) ensures that a user’s web browser will prevent HTTP requests being made to specified websites. This is different from a HSTS header as it occurs before a user has visited a website for the first time, provided they are using a web browser which supports the pre-load list. Thus, users are protected from attacks where an adversary prevents the HSTS header from being sent, such as in person-in-the-middle attacks.

The advice at \(https://hstspreload.org/\) should be followed to register websites for the HSTS Preload List.

Drop support for TLS 1.0 and TLS 1.1 immediately

All versions of TLS earlier than TLS 1.2, including SSL, are considered unsafe due to either having a flaw in the protocol itself or using vulnerable cipher suites that could leak confidential information. As such, support for versions of TLS earlier than TLS 1.2 will be dropped by Google\(^{39}\), Mozilla\(^{40}\), Microsoft\(^{41}\) and Apple\(^{42}\) starting in early 2020. Importantly, if ongoing support for TLS 1.0 or TLS 1.1 is required for any reason, organisations are strongly encouraged to contact the Australian Cyber Security Centre (ACSC) to discuss possible risk mitigation strategies.

---

\(^{33}\) [https://www.missingpadlock.com](https://www.missingpadlock.com)
\(^{35}\) [https://search.google.com/search-console/about](https://search.google.com/search-console/about)
\(^{36}\) [https://support.google.com/webmasters/answer/183668](https://support.google.com/webmasters/answer/183668)
\(^{38}\) [https://hstspreload.org/](https://hstspreload.org/)
\(^{42}\) [https://webkit.org/blog/8462/deprecation-of-legacy-tls-1-0-and-1-1-versions/](https://webkit.org/blog/8462/deprecation-of-legacy-tls-1-0-and-1-1-versions/)
Be aware of TLS 1.2 limitations

TLS 1.2 or higher is required for HTTP Version 2 (HTTP/2), which provides significant performance improvements over HTTP and should be supported in order to provide a better user experience. However, due to deficiencies in TLS 1.2 and some of the included cipher suites, use of HTTP/2 with TLS 1.2 is limited. For example, HTTPS may fail if non-secure renegotiation and compression has not been disabled for TLS 1.2 in web server settings. Further advice on limitations associated with the use of TLS 1.2 are available from the HTTP/2 standard\(^{43}\).

Begin transitioning from TLS 1.2 to TLS 1.3

As noted earlier, TLS 1.3 provides many security and performance benefits over TLS 1.2 and earlier versions. A simple way to disable older versions of TLS is to update web server configurations (e.g. an example configuration for Apache HTTP Server would be to include \texttt{SSLProtocol TLSv1.3} in the configuration file). In some cases, web server software will need to be updated to the latest version to support TLS 1.3. In addition, if using a Content Distribution Network (CDN), enabling TLS 1.3 for external clients may only require ticking a box in configuration settings. The encryption configuration between any origin servers and the CDN may also require adjustment.

It is important to note though, as TLS is not backwards compatible, dropping support for TLS 1.2 will result in legacy web browsers being unable to access websites using TLS 1.3. In such cases it is recommended that users be encouraged to update their web browser to the latest version to receive support for TLS 1.3 and to protect their confidentiality. However, if transitioning from TLS 1.2 to TLS 1.3 is not possible in the short term, risks to users can be reduced by:

- removing support for compression
- disabling client-initiated renegotiation
- disabling renegotiation if the version of TLS does not support secure renegotiation
- only enabling secure ciphers.

The acceptable cipher suites for each version of TLS are found in Annex A.

Mozilla’s wiki provides advice on TLS configuration, including example configurations for security, compatibility and legacy systems\(^{44}\). It also specifies what web browsers a particular configuration will be compatible with.

Additional resources

Additional resources on the implementation of TLS are available from the following sources:

- SSL Labs have advice on \textit{TLS Deployment Best Practices}\(^{45}\)
- the Internet Engineering Task Force have published \textit{Recommendations for Secure Use of TLS}\(^{46}\)
- Google’s Lighthouse tool\(^{47}\) can assist with locating HTTP-only content and also provides general tips on website security
- Mozilla has a TLS configuration generator for different web server software\(^{48}\)
- Mozilla Observatory provides a HTTP and TLS score card for websites\(^{49}\).

\(^{43}\) https://tools.ietf.org/html/rfc7540
\(^{44}\) https://wiki.mozilla.org/Security/Server_Side_TLS
\(^{46}\) https://tools.ietf.org/html/rfc7525
\(^{47}\) https://developers.google.com/web/tools/lighthouse
\(^{48}\) https://ssl-config.mozilla.org/
\(^{49}\) https://observatory.mozilla.org/
Case study using Let’s Encrypt and Certbot

This case study shows how to enable HTTPS using Let’s Encrypt and Certbot on Ubuntu 16.04 with Apache HTTP Server. It uses a simplified deployment scenario where the web server will perform its own TLS termination and generate its own certificate renewal requests. Depending on the security context, organisations may make use of proxies and other security infrastructure as part of their solution.

Firstly, add the required software repositories and install Certbot by running the following commands:

- `sudo apt update`
- `sudo apt install software-properties-common`
- `sudo add-apt-repository universe`
- `sudo add-apt-repository ppa:certbot/certbot`
- `sudo apt update`
- `sudo apt install certbot python-certbot-apache`

Once everything is installed, obtain a certificate by running the following command, `sudo certbot --apache --rsa-key-size 2048 --redirect --hsts` where:

- `--rsa-key-size 2048` sets the bit length of the RSA key to 2048
- `--redirect` automatically makes the website redirect to the HTTPS version
- `--hsts` makes sure the HSTS header is automatically sent.

There is also a `--uir` flag that will attempt to change every HTTP resource on the website to HTTPS. This is potentially dangerous as some resources might not have a HTTPS address, but if all resources on the website can be loaded securely this is a quick way to automate the transition of resources from HTTP to HTTPS.

After running the above certificate generation command, Certbot will ask for an email address for renewal and security notices. Type in the preferred email address and press Enter. Next, it will ask for agreement to their Terms of Service. After reviewing, type ‘A’ and press Enter. Then, it will offer to send emails from the Electronic Frontier Foundation. Press ‘Y’ or ‘N’ and press Enter. Finally, Certbot will ask for the website’s domain. Type it in and press Enter.

As certificates only last 90 days, certificates will need to be renewed often. Luckily, the Certbot packages come with a Cron Job that will renew certificates automatically before they expire. To test that it is all working correctly, run the following command, `sudo certbot renew --dry-run`. Note, the extra flags used when creating the certificate don’t have to be specified as ‘renew’ will use the settings of the last successfully created/renewed certificate.

Further information


Supporting guidance on website security, such as using Content Security Policy, HSTS and Frame Options, is available in the Protecting Web Applications and Users publication at https://www.cyber.gov.au/publications/protecting-web-applications-and-users.
Contact details

Organisations or individuals with questions regarding this advice can contact the ACSC by emailing asdassist@defence.gov.au or calling 1300 CYBER1 (1300 292 371).
Annex A: Acceptable cipher suites

This annex contains a list of acceptable cipher suites, including:

- cipher suites that fully align with the ISM
- cipher suites that partially align with the ISM
- cipher suites which organisations may wish to use subject to a risk management decision about their users’ needs and situation.

Ciphers are listed in order of preference (i.e. web servers should prefer ciphers in order going from top to bottom).

All cipher suites below are listed in their Internet Assigned Numbers Authority names. Different web server software often uses different syntax. Websites are available to assist in translating cipher suite names.\(^5\)

**Cipher suites for TLS 1.3**

The following cipher suites are recommended for TLS 1.3:

- TLS_AES_256_GCM_SHA384
- TLS_AES_128_GCM_SHA256
- TLS_AES_128_CCM_8_SHA256
- TLS_AES_128_CCM_SHA256.

Organisations may wish to make a risk management decision on the use of TLS_CHACHA20_POLY1305_SHA256. This cipher suite may provide better performance on platforms which do not have Advanced Encryption Standard (AES) hardware support, such as ARM-based devices like mobile phones, tablets and low-end laptops. However, while there are no publically disclosed weaknesses with this cipher suite it has not been subjected to the same standard of review and analysis as the recommended cipher suites.

**Cipher suites for TLS 1.2**

The following cipher suites are recommended for TLS 1.2:

- TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
- TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
- TLS_DHE_DSS_WITH_AES_256_GCM_SHA384
- TLS_DHE_RSA_WITH_AES_256_GCM_SHA384
- TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
- TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
- TLS_DHE_DSS_WITH_AES_128_GCM_SHA256
- TLS_DHE_RSA_WITH_AES_128_GCM_SHA256
- TLS_ECDHE_ECDSA_WITH_AES_256_CCM
- TLS_DHE_RSA_WITH_AES_256_CCM

\(^5\) [https://ciphersuite.info/](https://ciphersuite.info/)

---

\(^5\) [https://ciphersuite.info/](https://ciphersuite.info/)
- TLS_ECDHE_ECDSA_WITH_AES_256_CCM_8
- TLS_DHE_RSA_WITH_AES_256_CCM_8
- TLS_ECDHE_ECDSA_WITH_AES_128_CCM
- TLS_DHE_RSA_WITH_AES_128_CCM
- TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8
- TLS_DHE_RSA_WITH_AES_128_CCM_8.

Organisations may wish to make a risk management decision on the use of the following cipher suites as they may provide better performance on platforms which do not have AES hardware support, such as ARM-based devices like mobile phones, tablets and low-end laptops. However, while there are no publically disclosed weaknesses with these cipher suites they have not been subjected to the same standard of review and analysis as the recommended cipher suites:

- TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256
- TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256
- TLS_DHE_RSA_WITH_CHACHA20_POLY1305_SHA256
- TLS_ECDHE_ECDSA_WITH_CAMELLIA_256_GCM_SHA384
- TLS_ECDHE_ECDSA_WITH_CAMELLIA_256_GCM_SHA384
- TLS_ECDHE_RSA_WITH_CAMELLIA_256_GCM_SHA384
- TLS_ECDHE_RSA_WITH_CAMELLIA_256_GCM_SHA384
- TLS_ECDHE_ECDSA_WITH_CAMELLIA_128_GCM_SHA256
- TLS_ECDHE_ECDSA_WITH_CAMELLIA_128_GCM_SHA256
- TLS_ECDHE_RSA_WITH_CAMELLIA_128_GCM_SHA256
- TLS_ECDHE_RSA_WITH_CAMELLIA_128_GCM_SHA256
- TLS_DHE_DSS_WITH_CAMELLIA_256_GCM_SHA384
- TLS_DHE_DSS_WITH_CAMELLIA_256_GCM_SHA384
- TLS_DHE_RSA_WITH_CAMELLIA_256_GCM_SHA384
- TLS_DHE_RSA_WITH_CAMELLIA_256_GCM_SHA384
- TLS_ECDHE_ECDSA_WITH_CAMELLIA_128_GCM_SHA256
- TLS_ECDHE_ECDSA_WITH_CAMELLIA_128_GCM_SHA256
- TLS_ECDHE_RSA_WITH_CAMELLIA_128_GCM_SHA256
- TLS_ECDHE_RSA_WITH_CAMELLIA_128_GCM_SHA256
- TLS_DHE_DSS_WITH_CAMELLIA_128_GCM_SHA256
- TLS_DHE_DSS_WITH_CAMELLIA_128_GCM_SHA256
- TLS_DHE_RSA_WITH_CAMELLIA_128_GCM_SHA256
- TLS_DHE_RSA_WITH_CAMELLIA_128_GCM_SHA256
- TLS_DHE_RSA_WITH_CAMELLIA_128_GCM_SHA256
- TLS_DHE_RSA_WITH_CAMELLIA_128_GCM_SHA256.