CCN-CERT

BP/07

HTTPS Implementation Recommendations

BEST PRACTICE REPORT

JANUARY **2022**

ccn-cert centro criptológico nacional

CCN centro criptológico nacional

Edited by:

MINISTERIO DE DEFENSA
SUBSECRETARÍA DE DEFENSA
SECRETARÍA GENERAL TÉCNICA

CCN centro criptológico nacional

CENTRO CRIPTOLOGICO NACIONAL cn=CENTRO CRIPTOLOGICO NACIONAL, 2.5.4.97=VATES-S2800155J, ou=CENTRO CRIPTOLOGICO NACIONAL, o=CENTRO CRIPTOLOGICO NACIONAL, c=ES 2023.08.01 13:44:48 +02'00'

Paseo de la Castellana 109, 28046 Madrid

Date of issue: January 2022

[1] Raúl Siles, founder and security analyst at DinoSec, has been involved in the development and modification of this document and its annexes.

Index

Index

1. About CCN-CERT

The CCN-CERT (www.ccn-cert.cni.es) is the Computer Emergency Response Team of the National Cryptologic Centre, CCN (www.ccn.cni.es). This service was created in 2006 as the Spanish **Governmental/ National CERT** and its functions are set out in Law 11/2002 regulating the National Intelligence Centre, RD 421/2004 regulating the CCN and RD 3/2010, of 8 January, regulating the National Security Framework (ENS), modified by RD 951/2015 of 23 October.

According to all of them, the CCN-CERT is responsible for managing cyber-incidents that affect **public sector systems, companies and organisations of strategic interest** for the country and any classified system. Its mission, therefore, is to contribute to the improvement of Spanish cybersecurity, being the national alert and response centre that cooperates and helps to respond quickly and efficiently to cyberattacks and to actively address cyberthreats.

The CCN-CERT is the Computer Emergency Response Team of the National Cryptologic Centre, CCN

2. Executive overview

The proliferation of web technologies in recent years, together with the increase in their capabilities, functionalities, features and possibilities of use, allowing the performance of practically any management or task of daily life, both personal and professional, makes it necessary to consider the importance of making use of web communications as secure as possible, i.e. based solely on the use of the HTTPS protocol, which provides both authentication and encryption and integrity mechanisms.

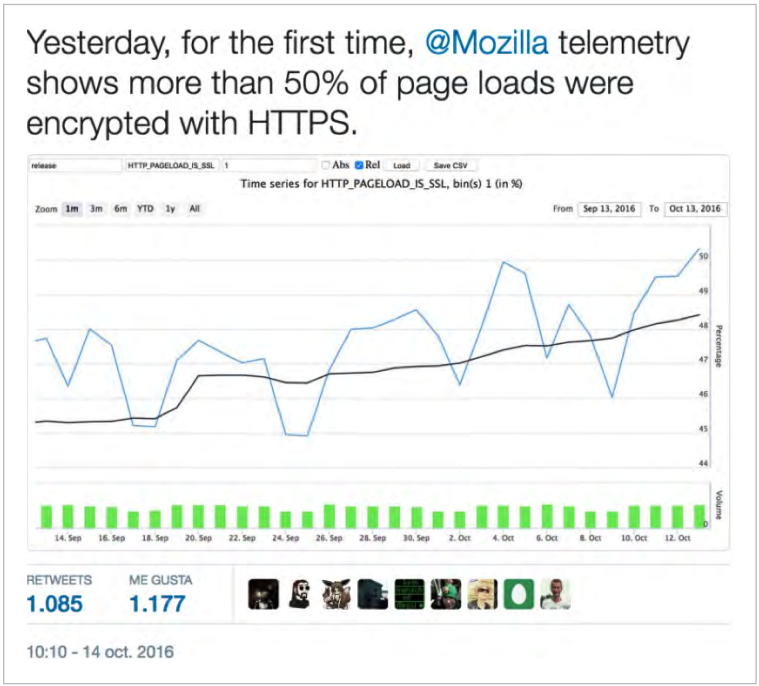
It is therefore necessary to promote the use of web technologies with a higher level of security through HTTPS, as opposed to HTTP, both within corporate environments and for private use, and regardless of whether they are accessed from traditional computers or from mobile devices, or from any other technological device, such as those associated with the Internet of Things (IoT, Internet of Things).

The evolution of HTTPS in the industry has seen very significant advances over the last few years, especially in 2016 and 2017 [Ref - 2], where for the first time in history, telemetry data collected by Mozilla shows that in October 2016 global HTTPS Internet traffic surpassed HTTP traffic (more than 50%). This upward trend in HTTPS usage over HTTP continues throughout 2017, reaching almost 55% in March[2].

The proliferation of web technologies in recent years, makes it necessary to consider the importance of making use of web communications as secure as possible, based solely on the use of the HTTPS protocol

[2] https://ipv.sx/telemetry/general-v2.html?channels=release&measure=HTTP_PAGELOAD_IS_SSL& target=1&absolute=0&relative=1

Figure 1.
Global Internet traffic according to Mozilla: HTTPS vs. HTTP. Source: Mozilla[3]



One of the main reasons why HTTPS should be used for access to web environments, servers and applications are the additional security features provided by this protocol used to establish secure communications, where the user can confirm that he is connecting to the desired web environment (authentication and identification) and the advantage that all traffic exchanged between the user and the web environment is encrypted, ensuring the integrity of the data transmitted, and preventing the traffic from being intercepted and manipulated by a third party.

Nowadays, all web traffic exchanged via the HTTP protocol should be considered sensitive, even when using public web servers and applications where authentication is not required, as user privacy and the protection of exchanged data must be ensured at all times. For this reason, it is necessary to generalise the use of the HTTPS protocol to all web environments, both public and private (or internal), of any organisation or company.

[3] https://twitter.com/0xjosh/status/786971412959420424, https://twitter.com/letsencrypt/status/786977436109934592

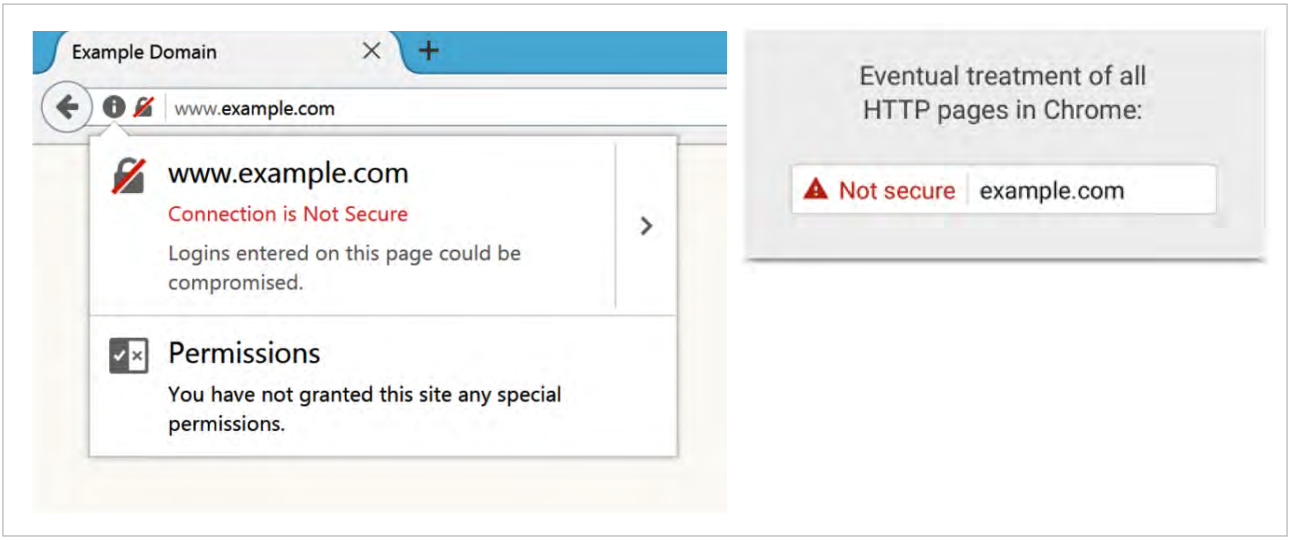


Figure 2.
Messages indicating that an HTTP web server is not secure.
Source: Firefox & Chrome

An example of this industry trend, from the point of view of widely used Internet services, materialised in October 2011, when Google converted its web browser to the HTTPS protocol[4], a publicly available service (where no authentication is required), but where personalised search terms used by users must be protected, ensuring their privacy. On the other hand, from the point of view of web clients, in January 2017 the main web browsers (e.g. Firefox 51[5] and Chrome 56[6]) have started to negatively highlight web services that do not make use of HTTPS and, especially, those that request sensitive information from the user, such as credentials or credit cards, highlighting them in red and with the website message "Not secure" (or "Insecure").

Despite the fact that web technologies are commonly used for personal and professional, private and relevant communications, for the exchange of sensitive information, and for access to numerous services, possibly due to their widespread use and ease of use, the level of perception of the real security threat of not using HTTPS has not been sufficiently high among end users and organisations. Organisations' web environments, and HTTP communications by users, are often subject to numerous attacks that, as a result, allow access to sensitive information, including personal data and even session IDs and credentials that would allow impersonation of their owners.

4 https://googleblog.blogspot.com.es/2011/10/making-search-more-secure.html

5 https://blog.mozilla.org/security/2017/01/20/communicating-the-dangers-of-non-secure-http/

6 https://security.googleblog.com/2016/09/moving-towards-more-secure-web.html

It should be noted that HTTPS only mitigates some specific attacks associated with web technologies, and additional security measures need to be put in place to protect web servers and applications against other attacks, such as SQL injection (SQLi), XSS (Cross-Site Scripting), CSRF (Cross-Site Request Forgery), session management attacks, LFI/RFI, etc.

Awareness of the exclusive use of HTTPS web environments by users and best practices in the implementation of HTTPS (design, configuration and maintenance) in web environments by their managers are the best defence to prevent and mitigate this particular type of web incidents and threats. The purpose of this document is to describe these practices in order to help those responsible for web environments to protect both their servers and web applications as well as end users, going deeper into the configuration and use of the currently existing protection mechanisms.

To this end, a detailed set of technical security guidelines and recommendations will be provided to improve the security and increase the protection capabilities of HTTPS implementation in web environments. Additionally, the section "ANNEX C. REFERENCES" provides several references and documentation to deepen all the security aspects described throughout this report[7], complementing the numerous references in the footnotes of the different sections of the report.

This Best Practices report, "HTTPS implementation recommendations" (CCN-CERT BP-01/17), due to its more technical nature, is aimed at a slightly different target audience than the other CCN-CERT Best Practices reports [Ref - 1], whose target audience is the end user of the technologies analysed. In this case, the target audience is the administrators and/or technical security managers of the environments, servers and web applications that must evaluate and implement support for the HTTPS protocol in the most secure way possible.

Awareness of the exclusive use of HTTPS web environments by users and best practices in the implementation of HTTPS (design, configuration and maintenance) in web environments by their managers are the best defence to prevent and mitigate this particular type of web incidents and threats

[7] Of particular note is the book "Bulletproof SSL and TLS" [Ref - 10].

3. Best practices and recommendations for the implementation of HTTPS

This report provides technical details for the secure implementation and deployment of the HTTPS protocol, based on TLS (Transport Layer Security), in web environments, servers and applications. In order to make it easier for the reader to understand the rationale for the various security recommendations described below, in some cases a brief description of, or references to, the security threats and incidents affecting HTTPS implementations in web environments today, and some of the techniques most commonly used by attackers, will be provided.

The set of recommendations shown is divided into multiple sections, each of them related to different elements, capabilities and functionalities associated with HTTPS implementations, including access through TCP/IP ports associated with web services, key generation, digital certificate management and other considerations associated with the Internet's Global Public Key Infrastructure (PKI), SSL and TLS protocols and their different versions, algorithms and crypto suites, the management of digital certificates and other considerations associated with the Internet's global public key infrastructure (PKI), SSL and TLS protocols and their different versions, cryptographic algorithms and suites, digital certificate revocation mechanisms, recommendations for publishing

web content and deploying web applications via HTTPS, etc. At the end of each section there is a summary of the main recommendations.

The aim of these recommendations is that the administrator and/or technical security manager of web environments, servers and applications can increase the level of protection and robustness of web services in relation to the implementation and support available for the HTTPS protocol, both from the point of view of its design and configuration, as well as its daily management and maintenance, thus preventing these web services from falling victim to known attacks on HTTP and HTTPS. Furthermore, the possible implications of how the proposed security mechanisms may affect the communication performance of the web environment will be taken into account at all times and recommendations will be provided to improve the performance of web traffic even when using HTTPS.

The aim of these recommendations is that the administrator and/or technical security manager of web environments, servers and applications can increase the level of protection and robustness of web services

It should be taken into account that some of the characteristics described and, therefore, of the security recommendations put forward, are dependent on the type of operating system used by the web environment (Linux, BSD, Unix, Windows, etc.), the type of web server used (Apache, nginx, IIS, etc.), the type of application server (Weblogic, Websphere, JBoss/WidFly, Tomcat, etc.), as well as the versions of all these elements, and the existence of other perimeter components, such as load balancers, content delivery networks (CDN, Content Delivery Network, reverse proxies, etc.), as well as the versions of all these elements, and the existence of other perimeter components, such as load balancers, Content Delivery Networks (CDNs), reverse proxies, firewalls, Web Application Firewalls (WAFs), HTTPS hubs or terminators, etc. Therefore, not necessarily all recommendations given will apply to all components in the web environment. In any case, it is recommended to apply as many recommendations as possible.

The references and more specific technical configuration examples reflected throughout this report make use of OpenSSL[8], as potentially the most commonly used (open source) library implementing the SSL and TLS protocols, and Apache[9], as the most widely used (also open source) web server on active websites on the Internet today and over the last few years (with almost 46% market share in February 2017[10]). In 2022 the situation has changed, with Apache dropping to 23.9% and nginx rising to 26.6%, currently leading in terms of market share. Therefore, administrators and/or technical security managers of web

8 https://www.openssl.org

9 https://www.apache.org (https://httpd.apache.org)

10 https://news.netcraft.com/archives/2017/02/27/february-2017-web-server-survey

environments, servers and applications that have to evaluate and implement support for the HTTPS protocol should identify in detail, in each case, the existing possibilities and the specific configuration particularities of the web technologies used in the environment where this report will be applied.

From a security point of view, it is recommended that all of the security mechanisms described above be implemented simultaneously and in a complementary manner, applying the principle of defence in depth. It should be taken into account that not all browsers and web clients have support for the different security standards and mechanisms suggested. Therefore, the application of multiple complementary measures, and although they may sometimes seem repetitive (such as, for example, the use of a 301 redirection from HTTP to HTTPS, the use of the HSTS header or the use of the CSP header to update insecure HTTP requests to HTTPS requests), will allow the web environment to be adapted to the capabilities available in the different clients that will access it.

Once all or most of the recommendations suggested throughout this report have been implemented, it is recommended to verify the overall status of HTTPS implementation for public web environments, servers and applications using the Qualys SSL Labs "SSL Server Test" [Ref - 4] service. When using the service, it is recommended to select the option *"Do not show the results on the boards"* before performing the analysis, in order not to appear in the public list of the most recently analysed websites, or with better or worse results[11]. The focus of the analysis should be on obtaining a final grade of A or A+. Any lower grade should be considered an anomaly or weakness, and should be addressed. It should be borne in mind that a grade of A today may be associated with a grade of B in a few months' time, as HTTPS security mechanisms and the minimum requirements associated with good practices are continually evolving, improving and being revised [Ref - 21].

[11] It should be noted that nothing prevents a third party from providing the address (or *hostname*) of an own web server through this service, without selecting this option, so it would be listed in the different categories mentioned, publicly available.

3.1. Access to TCP/IP ports associated with web services

Web services are associated by default to port TCP/80 in the case of the HTTP protocol, and to port TCP/443 in the case of the HTTPS protocol, which additionally makes use of authentication and encryption mechanisms using TLS.

When deploying a web service, and following the industry trend focused on converting and basing the web only on the HTTPS protocol (eliminating as much as possible the use of the HTTP protocol), it is recommended to make exclusive use of HTTPS, so it is necessary to have the TCP/443 port open in the web environment.

However, browsers and web clients currently use the HTTP protocol and TCP/80 port by default when the protocol to be used is not specified ("http://" or "https://"), for example, when the user only provides the name of the web server to which he/she wishes to connect (e.g. www.domain.com)[12].

In order to provide the highest possible accessibility and availability of the web environment, it is recommended to have the TCP/80 port also open[13], but configuring the web server or application to perform an automatic redirection (type 301 or 302) from HTTP to HTTPS for any request (connection attempt) on the TCP/80 port. Specifically, it is recommended to use a 301 ("301 Moved Permanently") redirection from HTTP to HTTPS[14], as this type of response can be cached and continuously applied by clients and web browsers (see section "3.7.3. Forcing the use (by default) of HTTPS").

Web services are associated by default to port TCP/80 in the case of the HTTP protocol, and to port TCP/443 in the case of the HTTPS protocol

[12] Only when, in a hypothetical future, web browsers use the TCP/443 (HTTPS) port by default, instead of HTTP, will it be possible to consider closing the TCP/80 port with high accessibility guarantees.

[13] https://scotthelme.co.uk/why-closing-port-80-is-bad-for-security/

[14] https://support.google.com/webmasters/answer/6073543?hl=en

Alternatively, a more restrictive implementation could be used where the TCP/80 port remains closed or filtered, and only TCP/443 port access is available. In this case, for reachability, all references to the web server must specify the HTTPS protocol via "https://", e.g., all links in "www.domain.com" pointing to "https://site.domain.com". Otherwise, if the TCP/80 port is closed and a user simply types the address (URL) of the web server or web application in the address bar of his web browser, he will get a connection error. If you do not have control over all external references pointing to the web server (a common situation when there are third party references pointing to it), or in case of doubt, it is recommended to open the TCP/80 port and configure the automatic redirection from HTTP to HTTPS previously described.

This scenario where it is recommended to have the TCP/80 port open, with an automatic redirection from HTTP to HTTPS, should also be used even in more secure scenarios where HSTS is used, and even if the domain has been added in the preloaded list by means of the "preload" directive (see section "3.7.4. Forcing strict (default) use of HTTPS: HSTS"), as situations may arise where a client does not make use of the HSTS preloaded list, or uses a version of the web browser that did not yet include the domain under study in the preloaded list.

3.1.1 TCP/IP port access recommendations for web services

The summary of TCP/IP port access recommendations associated with web services includes:

- **Have the standard TCP/443 port, associated with the HTTPS protocol, open.**
- **Have the TCP/80 port also open,** but configure the web server or application to perform an automatic 301 ("301 Moved Permanently") redirection from HTTP to HTTPS for any request on TCP/80 port.

3.2. Generation of the cryptographic keys

The correct generation of the cryptographic private keys associated with the digital certificates used in HTTPS, and specifically the proper selection of the key generation algorithm and key size (or length), is fundamental for HTTPS security. It should be noted that the weakest point related to keys is often associated with key management and the day-to-day task of keeping private keys secret.

The cryptographic keys, and their binding to the name of the entity represented by the web server through a digital certificate, constitute the cryptographic identity of the web server in HTTPS.

The most widely used algorithm today for the generation of private keys, and their associated public keys, is RSA (Rivest, Shamir and Adleman). It is recommended to use RSA keys with a minimum key length of 2,048 bits (providing 112 bits of security). The use of larger keys, e.g. 3072 bits can have a significant impact on the performance of the web environment.

In the future, the use of ECDSA (Elliptic Curve Digital Signature Algorithm) keys over RSA, sometimes referred to as ECC (Elliptic Curve Crytography)[15], is likely to become more widespread as they provide better performance than RSA for keys of equivalent length and security level. It is recommended to use ECDSA keys with a minimum key length of 256 bits (which provide 128-bit security and twice the speed of a 2,048-bit RSA key, and about 6 times the speed of an equivalent 3,072-bit RSA key).

The cryptographic keys, and their binding to the name of the entity represented by the web server through a digital certificate, constitute the cryptographic identity of the web server in HTTPS

[15] https://blog.cloudflare.com/a-relatively-easy-to-understand-primer-on-elliptic-curve-cryptography/

A balance between security and performance should always be sought, avoiding overdoing it by introducing "too much" security, which is why, at present, for standard web environments, it is recommended to use 2,048-bit RSA keys and 256-bit ECDSA keys. From a security and performance point of view, ECDSA is preferable to RSA.

A balance between security and performance should always be sought, avoiding overdoing it by introducing "too much" security

For reference, a 2,048-bit RSA key would be equivalent to a 224-bit ECDSA key (both with 112 bits of security), a 3,072-bit RSA key to a 256-bit ECDSA key (both with 128 bits of security), a 7,680-bit RSA key to a 384-bit ECDSA key (both with 192 bits of security), and a 15,360-bit RSA key to a 512-bit ECDSA key (both with 256 bits of security)[16]. The number of security bits would in turn be equivalent to the recommended bit size for keys used in symmetric cryptography (e.g. AES), and the number of ECDSA bits equivalent (approximately) to the length of the recommended hash types for similar robustness (e.g. ECDSA 256-bit and SHA-256)[17].

The main drawback of ECDSA today is that not all browsers and web clients (user agents) support it, especially some older ones (e.g. IE <= 8 or Chrome on Windows XP, or OpenSSL < 1.0.0). It is possible to evaluate the cryptographic algorithms or suites supported by a specific web client, including support for RSA and/or ECDSA keys, through the Qualys SSL Labs "SSL Client Test" service [Ref - 3], by selecting individual web clients and browsers to check all their HTTPS-related features [Ref - 6], or through "SSL Cipher Suite Details of Your Browser" [Ref - 5].

It is possible to use RSA and ECDSA keys simultaneously, in order to provide the highest possible level of security for each type of web client. However, this possibility depends on the technologies and platforms used in the implementation of HTTPS, so it is recommended to verify the existence of support for the simultaneous use of different keys in the web environment where this report will be applied. For example, Apache allows this type of configuration, using different RSA and ECDSA keys, and therefore different digital certificates, simultaneously.

In selecting and generating keys, it is necessary to assess which options are reasonably secure at present, and which will be reasonably secure until the key is replaced, as they will allow determining the period of time for which the keys will remain reasonably secure before they are replaced.

[16] https://casecurity.org/2014/06/10/benefits-of-elliptic-curve-cryptography/

[17] http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57pt1r4.pdf

It should be borne in mind that in most attacks and security incidents related to cryptography, encryption is bypassed rather than broken or compromised, as this is a much simpler task. Therefore, proper key management is recommended, keeping private keys secret at all times, making use of a good pseudo-random number generator (PRNG), protecting the keys with a strong password (*passphrase*) and storing them securely (e.g. using a Hardware Security Module (HSM), making secure backups of the keys, and renewing the keys periodically.

Under no circumstances should the process of generating the private key be delegated to the Certification Authority (hereinafter, CA) and, instead, it must be generated locally and provide the CA with a Certificate Signing Request (CSR), so that the CA can issue the associated digital certificate. The CSR must be generated using at least SHA-256 as signature or *hashing* algorithm. In addition, the principle of least privilege must be applied, exposing the private key to as few systems and persons as possible.

In order to ensure that the random generation of the numbers associated with the cryptographic keys is as good as possible, and to minimise their exposure, it is recommended to use *offline* equipment that is not exposed to any communications network, and with sufficient entropy (for example, not generating the key immediately after restarting the equipment, but after long periods of use), in order to generate sufficiently robust keys. Ideally the equipment should be FIPS (Federal Information Processing Standard) or Common Criteria compliant.

Key storage should be protected by using a strong password (*passphrase*) and, preferably, by using specific hardware security modules (HSM, Hardware Security Module), instead of having one or multiple copies of the keys in standard file systems on one or more servers. Likewise, the use of the same set of keys and their associated digital certificates on web servers with different security and criticality levels is very problematic, since a security incident on one of the servers would also expose the rest of the servers.

Key storage should be protected by using a strong password (*passphrase*) and, preferably, by using specific hardware security modules

Finally, keys should be properly managed, recording when they are generated, put into production and finally withdrawn or renewed, using a secure deletion process. Keys should be renewed periodically (and if possible automatically), so that they are not used for very long periods of time, and to minimise the associated impact in the event of a security incident or any other event that could expose them. After a security incident, previous digital certificates should be revoked, new keys should be generated, and new digital certificates should be requested to be issued for these new keys.

A common practice is to renew the cryptographic keys each time the associated digital certificate is renewed (see section "3.3.5. Renewal of digital certificates"), although the introduction of new security mechanisms such as HPKP (see section "3.3.9 Restriction of legitimate digital certificates: HPKP"), may influence the period and/or procedure for renewing cryptographic keys (as the pins are associated with the cryptographic key, and need to be updated if the key is renewed).

A common practice is to renew the cryptographic keys each time the associated digital certificate is renewed

3.2.1 Recommendations for the generation of cryptographic keys

The summary of recommendations for the generation of cryptographic keys includes:

- **Make use of RSA keys of 2,048 bits in length.**
- Alternatively, or in addition, **make use of ECDSA keys of 256-bit length,** as they offer better security and performance than RSA keys.
- It is recommended **to strike a balance between security and performance, avoiding overdoing it by introducing "too much" security,** so it is currently preferable to use ECDSA keys (256 bits) as opposed to RSA (2,048 bits).
- **Make simultaneous use,** if supported by the web technologies employed, **of ECDSA and RSA keys.**
- **Make use of a good random number generator to generate the keys.**
- **The storage of the keys must be protected by a strong password (*passphrase*)** and, alternatively, by specific hardware security modules (HSM, Hardware Security Module).
- **Proper key management:** keep private keys secret, have secure *backups* of keys, renew keys periodically, use a secure deletion process, renew keys after a security incident, expose private keys to as few systems and people as possible, etc.

3.3. Management of digital certificates

This section includes recommendations related to multiple aspects associated with the correct selection, obtaining and management of the digital certificates used by HTTPS and linked to the cryptographic keys previously mentioned (see section "3.2.1. Generation of keys"). Having robust and secure cryptographic keys and digital certificates prevents a potential attacker from easily spoofing the web environment.

Having robust and secure cryptographic keys and digital certificates prevents a potential attacker from easily spoofing the web environment

3.3.1 Types of digital certificates

There are currently three types of digital certificates, ordered by the level of security or trust they provide and their cost: Domain Validated (DV), Organisation Validated (OV) and Extended Validation (EV).

DV certificates are validated automatically and are the cheapest, while EV certificates apply standardised validation procedures (by the CAB Forum or CA/Browser Forum[18]), are more expensive, and are treated differently by web browsers, displaying the address bar (or URL) in green and reflecting information of the organisation owning the digital certificate next to the web address (or URL).

The use of one type of digital certificate or another depends on multiple factors, including economic factors as well as those associated with the organisation's image and the trust that it wishes to convey to the users of the server or web application.

[18] https://cabforum.org

3.3.2 Issuance of digital certificates: Certification Authorities (CAs)

This report focuses on the implementation of HTTPS on public web servers, so it is mandatory to have a digital certificate issued by a trusted and widely recognised public CA. It is therefore not possible, for the correct implementation of HTTPS, to make use of self-signed digital certificates or private CAs.

In the case of the National Security Framework (ENS), it is recommended to use a trusted service provider.

In the case of the National Security Framework (ENS), it is recommended to use a trusted service provider

Even for internal web servers, it is recommended to have a digital certificate issued by a trusted public CA or, failing that, by a private CA that is also widely recognised by web clients.

When requesting a digital certificate from a public CA, it is necessary to be aware that you are establishing a close medium to long term relationship with it, especially if you make use of new security mechanisms such as HPKP (see section "3.3.9. Restriction of legitimate digital certificates: HPKP").

Therefore, it is recommended to evaluate and select the CA based on objective criteria depending on the nature, criticality and security level of the web server where HTTPS is to be used. These criteria should include the CA's level of consolidation and maturity, its reputation in the industry, its recognition as a trusted CA in multiple browsers and web clients, the capabilities to manage numerous digital certificates, the type of administrative and technical support service provided by the CA, its position in the industry and its willingness to adopt new technologies and standards (such as CT – see section "3.3.6. Certificate Transparency (CT)" CAA – see section "3.3.7. CAA (Certification Authority Authorisation)" OCSP Stapling – see section "3.6.3. OCSP Stapling" the possibility of obtaining digital certificates based on RSA and ECDSA keys – see section "3.2. Generation of cryptographic", etc.), and of course, the level of security and protection mechanisms of the CA itself and its transparency when managing security incidents (based on the history of past incidents).

Another interesting element to consider during the evaluation of the CA to be selected is the digital certificate revocation mechanisms available (see section "3.6. Digital certificate revocation mechanisms"), offering at least CRL and OCSP, together with the analysis of the availability and performance offered for these services.

As an alternative to commercial CAs, there is currently a non-profit CA called Let's Encrypt [Ref - 11], which aims to generalise the use of HTTPS on the Internet by issuing free digital certificates (DV type). Let's Encrypt represents a valid alternative for obtaining legitimate and widely recognised digital certificates, and is characterised by using a model with a short certificate renewal period (currently set at 90 days, 3 months[19]) in order to minimise the impact of a possible security incident and to promote the automation of the digital certificate application and renewal process.

For reference[20], in February 2016 Let's Encrypt introduced support for the generation of digital certificates based on ECDSA keys[21], and by September 2017 announced the availability of a root CA and intermediate CAs also based on ECDSA keys, making a complete ECDSA-based certification (or trust) chain available.

In addition to the selection of the CA that will issue the digital certificates, once issued, the CA includes its own signature on the certificate to prove its validity. To do so, it makes use of a signature algorithm, and nowadays it is considered that only digital certificates signed using SHA-256, or more robust algorithms (e.g. SHA-384, SHA-512...), also referenced as SHA-2, should be available. Under no circumstances should SHA-1 or MD5 be used for signing the certificate.

There is currently a non-profit CA called Let's Encrypt, which aims to generalise the use of HTTPS on the Internet by issuing free digital certificate

[19] https://letsencrypt.org/2015/11/09/why-90-days.html

[20] https://scotthelme.co.uk/ecdsa-certificates/

[21] https://letsencrypt.org/upcoming-features/

Due to the fact that the request for a certificate via a CSR does not allow specifying the signature algorithm to be used by the CA, it is recommended to check with the CA, before requesting the digital certificate, which algorithms it is currently using for signing digital certificates. Additionally, it is also recommended to check with the CA what the complete certification (or certificate) chain associated with the requested digital certificate will be, i.e. the complete list of intermediate CAs associated with the issuance of the certificate from the certificate to the root CA, and to identify the root CA itself that will be assigned to the requested digital certificate.

It is important to clarify that the recommendation not to use SHA-1 applies to the signing of the CAs' own root digital certificates. The signature algorithms used in the past, such as MD5 or SHA-1, are considered insecure and should not be used to sign digital certificates:

- **MD5** was designed to provide 64-bit security and was completely breached, in a practical way over digital certificates used for HTTPS, in 2009[22].

- **SHA-1** was designed to provide 80 bits of security, although it actually provides 61 bits of effective security. Since January 2016 CAs should not issue digital certificates signed with SHA-1[23] (under initial request from Microsoft), and finally in January/February 2017 the main web browsers (Firefox 51[24], Edge & IE 11[25], or Chrome 56[26]) stopped supporting digital certificates signed with SHA-1 as valid.

  In February 2017 Google, together with the CWI Institute in Amsterdam, published a practical attack on SHA-1 [Ref - 12].

It is important to clarify that the recommendation not to use SHA-1 applies to the signing of the CAs' own root digital certificates

22 https://documents.epfl.ch/users/l/le/lenstra/public/papers/lat.pdf

23 https://threatpost.com/sha-1-end-times-have-arrived/

24 https://blog.mozilla.org/security/2016/10/18/phasing-out-sha-1-on-the-publicweb/

25 https://blogs.windows.com/msedgedev/2016/11/18/countdown-to-sha-1-deprecation/

26 https://security.googleblog.com/2016/11/sha-1-certificates-in-chrome.html

3.3.3 Entity associated with the digital certificate

Within the different fields or security elements included in a digital certificate (see section "3.3.4. Validity of the digital certificate"), it is worth highlighting the name or identifier of the entity (environment, server or web application) to which the certificate is associated.

When a browser or web client connects via HTTPS to the web server, it verifies that the name of the entity associated with the digital certificate corresponds to the name used in the web address (or URL) visited. Otherwise, certificate errors will be generated that may confuse users and reduce trust in the web environment.

It is therefore essential to ensure that the names and entities reflected in the digital certificate match and cover (represent) all the web servers that make use of this certificate, either with a direct or unique reference (such as www.domain.com), through a list with all the names of the web servers (for example, www.domain.com, portal.domain.com and site.domain.com), or through a wildcard digital certificate that covers all the subdomains of the main domain (such as *.domain.com)[27]. The main consideration when making use of a digital wildcard certificate is that it should only be used on web servers with the same level of security, to minimise the impact of a possible security incident (and access to the private keys linked to the digital certificate) and, again, the principle of least privilege should be applied, trying to reduce the need to share the digital certificate and private key between numerous departments, systems and individuals.

Although it is also possible to obtain multi-domain digital certificates, i.e. valid for web servers of different domains, for the same reason (to promote their use in environments with the same level of security and to reduce the need to share the digital certificate and the private key), it is recommended to separate the digital certificates for each domain.

Within the different fields or security elements included in a digital certificate, it is worth highlighting the name or identifier of the entity to which the certificate is associated

[27] The use of digital wildcard certificates is common in CDNs that manage HTTPS traffic to multiple web servers within the same organisation.

On the other hand, it is recommended to make sure that the digital certificate includes both the name of the web server represented (or the list of web servers, or the wildcard certificate) and that of its domain, i.e. www.domain.com and domain.com (without the "www" prefix), or *.domain.com and domain.com (in the case of using a wildcard digital certificate). This scenario is common for digital certificates issued today (but must be explicitly verified), as both names usually refer in the DNS service to the same IP address, and therefore to the same server. As a general rule, the digital certificate should be valid for all names existing in the DNS service for the associated web server.

It is recommended to make sure that the digital certificate includes both the name of the web server represented and that of its domain

In the past, the name of the entity associated with the digital certificate was reflected in the "CN" (Common Name or commonName) field of the certificate. Subsequently, RFC 2818 [28] suggested identifying the name through the "SAN" field (Subject Alternative Name, extension "subjectAltName" of type "dNSName") and, alternatively, in its absence, through the "CN". This RFC, published in May 2000, already prioritised the "SAN" over the "CN", undervaluing the latter. Despite this, numerous web clients have supported both fields over the last few years.

However, from Firefox 48[29] and Chrome 58[30] there is no support for the identification of the name via the "CN", generating a certificate error in case the expected web server name is not found in the "SAN" field. It is therefore recommended to use only the "SAN" field to reflect the list of names of the entities associated to (or represented by) the digital certificate.

[28] https://tools.ietf.org/html/rfc2818

[29] https://bugzilla.mozilla.org/show_bug.cgi?id=1245280

[30] https://www.chromestatus.com/features/4981025180483584

3.3.4 Validity of the digital certificate

X.509 digital certificates have numerous fields or security elements that are evaluated to determine the validity of the certificate by a browser or web client. For this reason, it is necessary to ensure at all times that all their values are valid, including:

- **Name of the entity or web server** (see section "3.3.3. Entity associated with the digital certificate").
- **Period of validity or expiry of the certificate** (with the start and end date of this period).
- **CA that has issued and signed the certificate** (see section "3.3.2. Issuing digital certificates: Certification Authorities (CAs)".
- **Revocation of the certificate** (see section "3.6. certificate revocation mechanisms Digital ").
- **Purpose of the certificate:** server authentication (associated with the web servers covered in this report), client authentication, executable code or software signing, e-mail, etc. (this classification is clearly visible especially in Windows environments).

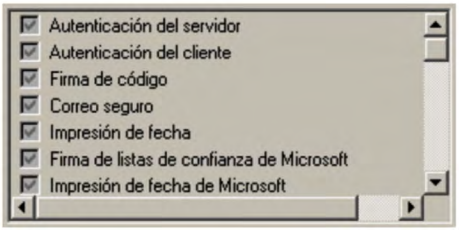


Figure 3.
Different purposes of using a digital certificate.
Source: Microsoft

More specifically, when assessing the validity of an X.509 digital certificate, it is also necessary to check that the entire certification chain is valid, and more specifically, that it is not incomplete. It is recommended that the web server provides web clients with the complete certification chain (thus preventing them from having to resolve and reconstruct it autonomously, a situation that can lead to errors), providing each and every digital certificate issued or associated to the CA in the correct order, i.e. from the intermediate CA immediately after the root CA, to the web server's own digital certificate. The certification chain should include only the necessary digital certificates; for performance reasons it is not recommended to add the root CA's certificate to the chain, as it should already be known by browsers and web clients (as it is available in their store of trusted CAs).

It is possible to evaluate the certification chain of the web server with web tools such as "Certificate Analyser"[31].

[31] https://report-uri.io/home/certificate_analyser/

3.3.5 Renewal of digital certificates

The usual standard period of validity or renewal of certificates is currently between 1, 2 or 3 years, depending on the type of certificate (DV, OV or EV). Logically, the period to be selected is influenced by the economic cost (with the CAs usually offering some type of discount for longer periods) and the frequency with which the associated administrative renewal procedures are to be carried out.

The usual standard period of validity or renewal of certificates is currently between 1, 2 or 3 years, depending on the type of certificate (DV, OV or EV)

It is recommended to renew digital certificates annually, and even more frequently if it is possible to automate the renewal process (see section "3.3.2. Issuing digital certificates: Certification Authorities (CAs)" and references to Let's Encrypt). If it is considered very complex to completely, effectively and reliably revoke a compromised digital certificate, from a practical point of view, digital certificates with a shorter lifetime would be considered more secure.

In March 2017, a voting process was launched in the CAB Forum (Ballot 193) to define a new standard stating that all digital certificates (issued as of 1 March 2018) will have a maximum validity period of 825 days[32] (approximately 27 months, i.e. 2 years and three months), as opposed to the current maximum period of 39 months (i.e. 3 years and three months). The result of the vote confirmed that the new maximum validity period of digital certificates issued as of 1 March 2018 is set at 825 days[33].

Digital certificates must be renewed periodically and always before they expire. Each time the digital certificate is renewed, it is also recommended to renew the complete certification chain, including all intermediate CAs (whose digital certificates may also have expired), in order to provide web clients with a complete and updated list of the certification chain. It is also recommended to renew the cryptographic keys each time the associated digital certificate is renewed, unless HPKP is being used (see section "3.3.9. Restriction of legitimate digital certificates: HPKP").

[32] https://cabforum.org/pipermail/public/2017-March/010024.html

[33] https://cabforum.org/pipermail/public/2017-March/010098.html

It is recommended to deploy the new digital certificate, once renewed, approximately one week after having obtained it, trying to avoid problems with web clients whose time reference is not correctly configured and to give the CA enough time to propagate the new certificate as valid in its OCSP service. Additionally, it should be taken into account that it is not advisable to revoke the previous digital certificate immediately after renewal, but rather that there is a certain overlap in the validity period of both certificates to allow a smooth migration process from one to the other.

3.3.6 Certificate Transparency (CT)

Google's Certificate Transparency (CT) standard [Ref - 13], or Certificate Transparency, establishes an open environment (or *framework*) that enables the supervision, monitoring and auditing (in the issuance process) of digital certificates by CAs, in near real-time. The initial proposal is becoming an industry-wide standard, associated with RFC 6962 [Ref - 14].

CAs have proven over the last few years to be vulnerable to the inappropriate use and manipulation of their digital certificate issuance procedures. CT makes it possible to identify the issuance of new certificates, as these must be reflected by the CAs in a publicly verifiable register, and whose integrity cannot be manipulated, as it makes use of cryptographic mechanisms that only allow records to be added (but not deleted or modified)[34]. CT allows for early identification of multiple undesired scenarios, such as when a CA issues a certificate in error, when a CA (which, for example, has been acquired by another CA) issues unsolicited certificates, or when a CA becomes malicious, or has been compromised, and issues certificates to impersonate certain websites. Early detection mitigates the impact of illegitimate certificates and, in turn, provides transparency and public scrutiny capabilities of the health and integrity of the entire Internet's global public key infrastructure (PKI) ecosystem.

Google's Certificate Transparency (CT) standard, or Certificate Transparency, establishes an open environment (or framework) that enables the supervision, monitoring and auditing of digital certificates by CAs, in near real-time

[34] Merkle hash trees" are used for this: www.certificate-transparency.org/log-proofs-work

Once the issuance of a non-legitimate digital certificate has been identified by TC, actions can be taken to revoke it as soon as possible (see section "3.6. Digital certificate revocation mechanisms") and, if necessary, replace it with a new legitimate digital certificate.

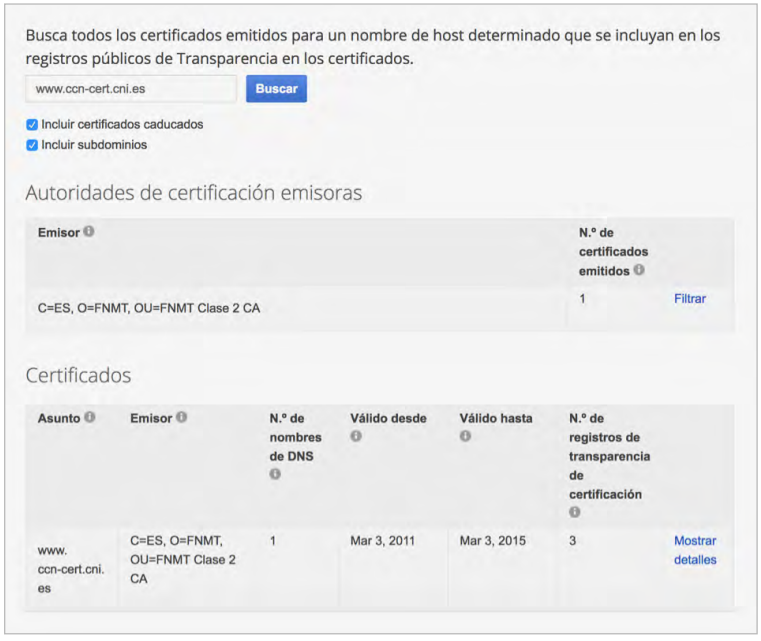
CT makes it possible to monitor the public logs and identify the issuance of digital certificates for its own domains without the domain owner's authorisation, either by other CAs other than the CA currently in use, or even by the CA itself. To do this, resources are available [Ref - 15][35] that allow for querying the existing records for a domain, or even for a specific name (or host), with the possibility of including in the query digital certificates that have already expired and those associated with subdomains. The response obtained makes it possible to identify the issuing CAs and the certificates registered publicly in CT.

For reference, on 13 December 2016, almost 172 million entries had been recorded in CT logs, while on 16 March 2017 that number had increased to almost 249 million entries.

Complementarily, there are CT monitoring services to be informed of changes in the digital certificates associated with a domain, such as Computest Suricat (https://suricat.io), or Cert Spotter (https://sslmate.com/certspotter/).

Once the issuance of a non-legitimate digital certificate has been identified by CT, it is possible to take action to revoke the certificate as soon as possible and, if necessary, replace it with a new legitimate digital certificate

Figure 4.
Results of the CT consultation for www.ccn-cert.cni.es.
Source: Google



[35] https://www.entrust.com/ct-search/

The Chrome web browser requires all EV-type certificates to be available in CT from 1 January 2015[36] and, due to irregularities by Symantec as a CA, Chrome 53 (from October 2015) requires the use of CT for all digital certificates issued by Symantec after 1 June 2016[37]. Additionally, as of October 2017, Chrome will require CT for all new digital certificates[38].

The CT environment is composed of three main components[39]:

- **Logs or certificate records:**
  Service that is responsible for maintaining the public cryptographic records with digital certificates, where only records can be added (usually provided by the CAs).
- **Monitors:**
  Service that connects to public registers in search of suspicious certificates.
- **Auditors:**
  Service that, on the one hand, verifies that public registries are working correctly and are cryptographically consistent and, on the other hand, can verify the existence of a specific certificate in the CT registry. Web browsers will eventually implement these capabilities.

HTTPS web servers can provide CT information to web clients in their responses, during TLS session establishment, using three methods: X.509v3 digital certificate extension (provided by the CA), TLS extension (available for example in Apache via the "mod_ssl_ct" module[40]) or OCSP Stapling[41] (also if supported by the CA).

36 https://sites.google.com/site/certificatetransparency/ev-ct-plan

37 https://security.googleblog.com/2015/10/sustaining-digital-certificate-security.html

38 https://groups.google.com/a/chromium.org/forum/#!topic/ct-policy/78N3SMcqUGw

39 http://www.certificate-transparency.org/what-is-ct

40 https://httpd.apache.org/docs/trunk/mod/mod_ssl_ct.html

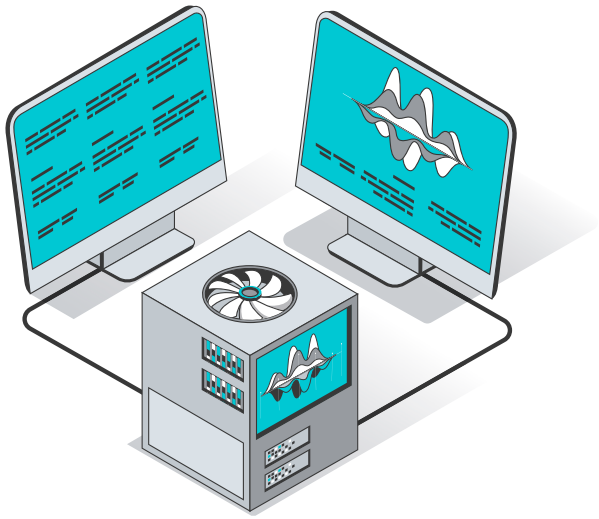
41 http://www.certificate-transparency.org/resources-for-site-owners

The simplest method to provide CT information from the web server is to make use of a CA that issues digital certificates including an X.509 extension within them that incorporates the *timestamp* of the signature or proof of the certificate's incorporation in the CT registry (SCT, Signed Certificate Timestamp)[42], evidencing its existence in the CT registry. However, it is preferable to make use of a new TLS extension created for this purpose or OCSP Stapling (if the CA provides support for including SCT in the OCSP response), since both methods allow including new SCT values for new records (adding to, or replacing, previous values) without the need to renew the digital certificate.

The simplest method to provide CT information from the web server is to make use of a CA that issues digital certificates including an X.509 extension

Additionally, in January 2017 Chrome proposed the creation of a new HTTP header, called "Expect-CT" [Ref - 16], which indicates that the web server has CT support and that the client should expect to receive a valid SCT value. As with other security HTTP headers such as HPKP (see section "3.3.9. Restriction of legitimate digital certificates: HPKP") or CSP (see section "3.7.6. Content Security Policy (CSP)"), "Expect-CT" can be used in an informative way by generating notifications from web browsers (to the reference specified by the "report-uri" directive), to detect the absence of valid SCTs, or in a strict way, by blocking HTTPS connections that are associated to a digital certificate that is not CT compliant. To do so, "Expect-CT" does not make use of two different HTTP headers, as in the case of HPKP or CSP, but uses the optional "enforce" directive within the same header.

As Chrome will require CT for all new certificates in October 2017, this header allows to protect also old certificates (issued before October 2017) or new certificates, but which have been issued by a malicious CA with previous dates.



[42] https://www.certificate-transparency.org/how-ct-works

[43] https://groups.google.com/a/chromium.org/forum/#!topic/blink-dev/tgn5R-58iek

3.3.7 CAA (Certification Authority Authorisation)

The Certification Authority Authorisation (CAA) standard [Ref- 17] establishes a new type of record in the DNS name resolution service (RR, resource record, of type 257) that allows the owner of a domain to specify that one or more CAs have the authority to issue digital certificates for that domain and that, therefore, any other CA is not authorised to issue them.

Therefore, CAA links the DNS service with the process of issuing digital certificates used, for example, in HTTPS, allowing the DNS service to be used as an external validation mechanism associated with the HTTPS protocol and, in particular, with the management and issuance of digital certificates.

CAA allows a (well-intentioned) CA to add additional (administrative) controls to reduce the risk of issuing digital certificates unintentionally and contrary to the stipulations of the web domain owner.

CAA allows a CA to add additional controls to reduce the risk of issuing digital certificates unintentionally and contrary to the stipulations of the web domain owner

In February 2017, a voting process was launched in the CAB Forum (Ballot 187) to make the CAA standard mandatory for CAs[44], whereby CAs must verify and check DNS CAA records before issuing a new digital certificate. The result of the vote ratified that the AAC check is mandatory for CAs[45], effective 8 September 2017 [Ref - 18]. In case the CAA configuration does not allow a CA to issue digital certificates for that domain, the issuance process must be interrupted (and potentially manually reviewed by CA staff).

These checks introduce significant improvements to the Internet's global public key infrastructure (PKI), composed of all trusted public CAs, preventing the ecosystem and trust in public CAs from being as weak as the weakest CA.

[44] https://cabforum.org/pipermail/public/2017-February/009722.html

[45] https://cabforum.org/pipermail/public/2017-March/009988.html

The CAA records provide granularity for determining the CAs associated with the issue, the CAs associated with the issuewild certificates, see section 3.3. Entity associated with the digital certificate"and provide notification capabilities in case of errors or invalid certificate requests ("iodef"), via a URL or via an e-mail address.

The format of the CAA records includes the domain associated with the digital certificates and which establishes the restriction in the DNS, and the domain name of the CA authorised to issue digital certificates for that domain[46]:

dinosec.com.	INCAA128	issue "letsencrypt.org"
dinosec.com.	INCAA128	issuewild "letsencrypt.org"
dinosec.com.	INCAA128	iodef "mailto:info@dinosec.com"

The CAA standard is sufficiently flexible and allows additional future properties to be defined by means of (variable) name and value pairs.

3.3.8 DNSSEC and DANE

DNSSEC (Domain Name System Security Extensions), is a set of technologies that add integrity to the domain name system, DNS (Domain Name System). Today, an attacker active on a network can intercept any DNS request and arbitrarily modify its response. With DNSSEC, all responses can be cryptographically traced back to the DNS root.

On the other hand, DNS-based Authentication of Named Entities (DANE) is a separate standard that builds on the foundation created by DNSSEC, and provides links between DNS and TLS. DANE could be used to augment the security of the existing CA-based PKI ecosystem or to provide a different alternative.

Although there is no consensus on whether DNSSEC is the correct address to which the Internet should be directed, support for it continues to improve. Browsers do not yet support DNSSEC and DANE, but there are indications that they are starting to be used to improve the security of the e-mail system.

[46] The value 128 adds a criticality flag for that DNS policy, which will allow adding new semantics in future versions of AAC, and indicate that the policy must be understood by the CA in order to be able to process the associated AAC records and proceed to issue the digital certificate.

3.3.9 Restriction of legitimate digital certificates: HPKP

The HTTP protocol security header known as HPKP or PKP (HTTP Public Key Pinning) [Ref - 19] allows the web server to declare that a digital certificate for that web server should only be considered valid if it is included in the specific list of certificates (pins) listed in that header. As a result, web browsers and web clients with HPKP support will not connect to the web environment if the received digital certificate (or any certificate in the certification chain, e.g. those of intermediate CAs) does not match the received pins.

This security header is deprecated and currently, in 2022, not recommended due to, among other factors, risks such as HPKP Suicide or RansomPKP. Some browsers may still support it for compatibility reasons. This header has been replaced by Certificate Transparency.

HPKP is probably one of the most stringent security mechanisms available today, and introduces the risk of blocking access to the web environment if not configured correctly, so it should be implemented with caution and careful evaluation of its implementation. In particular, it should be used if it is considered that there is a risk that the web environment could be spoofed by a fraudulent or non-legitimate certificate. For all the protection measures suggested throughout this report, it is recommended to evaluate the security benefit obtained against the difficulty or complexity of implementation and management[47]. However, HPKP can be implemented without any risk using only its notification capabilities (described below), which allows using browsers and web clients as sensors for the detection of spoofing attacks in case of identifying non-legitimate digital certificates.

HPKP is probably one of the most stringent security mechanisms available today, and introduces the risk of blocking access to the web environment if not configured correctly, so it should be implemented with caution and careful evaluation of its implementation

The objective of HPKP is to mitigate the risk associated with one of the main weaknesses of the Internet's global public key (PKI) ecosystem, the fact that any public CA (there are currently hundreds of them) can generate a valid digital certificate for any Internet web environment, and without the express authorisation of its owner. HPKP prevents, in the web client itself, MitM (Man-in-the-Middle) attacks[48] that make use of valid but not legitimate or authorised certificates, i.e. that

[47] https://wiki.mozilla.org/Security/Guidelines/Web_Security

[48] Mainly due to the compromise of a CA, or to malicious or anomalous behaviour on the part of a CA.

have been issued by any of the publicly recognised CAs, although their issuance has not been authorised by the web server owner. In fact, HPKP allows to remove the default trust in CAs and in the Internet's global public key infrastructure (PKI), and to trust only a very specific set of digital certificates (pins).

This security measure has been designed following the increase in security incidents involving CAs in recent years, with well-known cases such as DigiNotar or Comodo in 2011, or StartCom and WoSign later in 2016 (with multiple violations of the CAs' rules of the game), and which have allowed attackers (or CAs) to generate digital certificates for numerous non-legitimate websites, but perfectly valid for all browsers and web clients.

The HPKP header must be included in each and every HTTPS response generated by the web server or application:

```
Public-Key-Pins:
pin-sha256="<base64-A==>"; pin-sha256="<base64-B==>"; ...; max-age=5184000 [; includeSubDomains][; report-uri="<URL>"]
```

NOTE:
In the case of HPKP, as for CSP, it is possible to strictly and effectively enforce the policy while receiving violation reports (via "report-uri"). This option is very interesting to identify attacks using non-legitimate certificates.

It should be noted that the web browser enforces the use of HTTPS for a specific set of digital certificates for the period indicated by the "max-age" directive (specified in seconds) of the HPKP header. It is recommended to use a "max-age" value of 5184000 (60 days or 2 months), as recommended by the RFC, in order to strike a balance between security and availability.

To minimise the possible impact of an incorrect configuration when starting the HPKP deployment, it is recommended to progressively increase the "max-age" value, as suggested for the HSTS header (see section "3.7.4. Forcing strict (default) use of HTTPS: HSTS"). At the same time, accesses to the reporting URL indicated by the "reporturi" directive should be thoroughly monitored.

The "report-uri" directive refers to the web address where web browsers and clients should report pin validation errors, both in standard mode and in HPKP reporting mode, using the format defined by the RFC. It is recommended to use an HTTPS web reference associated to a web server other than the one that has sent the HPKP header, otherwise it will not be possible for the web browser to submit the report due to the pin error. It is possible to make use of your own specific web server, on the same or another domain, to receive these reports, or to make use of a third party service for this purpose, such as "Report URI"[49] [50].

The "report-uri" directive refers to the web address where web browsers and clients should report pin validation errors, both in standard mode and in HPKP reporting mode, using the format defined by the RFC

From a security point of view, in the case of specifying pins associated to wildcard certificates or to root or intermediate CAs, for simplicity, HPKP should be implemented for the whole domain, and not only for individual web servers. To do so, the "includeSubDomains" directive must be used.

In addition, as for HSTS (see section "3.7.4. Forcing strict (default) use of HTTPS: HSTS"), it is important to highlight that HPKP is a TOFU (Trust On First Use) security mechanism, i.e. it does not apply the first time the web client connects to the web server. In the case of HPKP, however, there is no public preloaded list as is the case with HSTS. Internally, web browsers do have a static list or set of pins (static_spki_hashes), known as "Static Public Key Pinning", which applies to some domains associated with the companies linked to the web browsers themselves (Google, Firefox, etc.) or, exceptionally, to other large companies and domains included by them (Facebook, Twitter, etc.)[51].

Additionally, as with other security HTTP headers such as CSP (see section "3.7.6. Content Security Policy (CSP)"), and again to minimise the possible impact of incorrect configuration when starting the HPKP deployment, it is recommended to start by making use of the HPKP reporting capabilities via the extended "...-Report-Only" header and the "report-uri" directive. This header should be tested initially to identify the correct functioning of the policy. If a browser or web client, with HPKP Report-Only support[52] (such as Chrome 46 or Opera 33[53]), receives a non-legitimate digital certificate, i.e. one that is not included in

[49] https://report-uri.io

[50] https://scotthelme.co.uk/report-uri-io-new-features-new-reports-new-tools/

[51] https://community.qualys.com/thread/17152-what-is-static-public-key-pinning

[52] https://developer.mozilla.org/es/docs/Web/HTTP/Headers/Public-Key-Pins-Report-Only

[53] No soportada aún en Firefox: https://bugzilla.mozilla.org/show_bug.cgi?id=1091177

the HPKP header pins, it will not block HTTPS traffic (as is the case with the standard HPKP header, which enforces the HPKP policy without exception), but will generate a report directed to the URL indicated in the header by the "report-uri" directive. In this scenario, effectively, web clients are used as sensors for detecting web spoofing attacks:

```
Public-Key-Pins-Report-Only:
pin-sha256="<base64-A==>"; pin-sha256="<base64-B==>"; report-uri="https://
dominioinformes.es/hpkp-report"
```

NOTE:
In case of using "...-Report-Only" it is not necessary to use the "max-age" directive.

It is recommended to start using the HPKP header only on a specific web resource, until its correct configuration is verified, in order to avoid receiving a multitude of violation reports or pin validation errors.

Two key elements to consider when implementing HPKP (described below) are how to generate the pins for existing digital certificates, and on which digital certificate in the current certification chain the pin reflected in the HPKP header should be associated.

When creating pins, or references to legitimate digital certificates, there are theoretically two options: refer directly to the digital certificate, or to the public key included in the digital certificate. HPKP references the public key, specifically the "Subject Public Key Info" (SPKI) field of X.509 digital certificates, to avoid relying on the renewal and other management tasks of digital certificates. The SPKI structure includes the public key together with certain metadata, such as the cryptographic algorithm used for its generation (e.g. RSA).

HPKP verification in browsers or web clients therefore requires that the actual certification chain presented to the browser or web client by the web environment, server or application must include at least one SPKI structure whose fingerprint matches the fingerprint of one of the pins in the HPKP hea der.

The HPKP header must include at least two pins (reflected above as A and B): a primary pin, linked to the SPKI value of one of the digital certificates in the current certification chain, and a backup *pin*, which must not be in the current certification chain. The backup pin can be used, for example, to renew the cryptographic key (public and private) of the server, and its associated digital certificate, after a security incident where the main pin is compromised. Additionally, it is possible to configure more pins in the HPKP header, both main and backup pins.

The value reflected by the HPKP header in the "pin-sha256" directive, associated to the pin value, corresponds to a SHA-256 hash of the SPKI field value in base64. The presentation "HTTPS: TLS, not SSL" [Ref - 2] provides the necessary details to generate both the primary and backup pins using OpenSSL.

From the point of view of the management and renewal of cryptographic keys (see section "3.2. Generation of cryptographic keys"), it should be noted that it is common practice to renew the cryptographic keys each time the associated digital certificate is renewed (see section "3.3.5. Renewal of digital certificates", as is the default for the Let's Encrypt CA). However, because the HPKP pins are associated to the cryptographic key (or public key), it would be necessary to update them if the key is renewed. In this case, it is recommended not to renew the cryptographic key during the standard digital certificate renewal process, so that the value of the existing pins can be maintained.

When associating the pin reflected in the HPKP header to a digital certificate of the current certification chain, there are several possibilities, the most restrictive being the one that prefers to create a pin for the public key (and, consequently, the final digital certificate) of the web server, and the most permissive one that associates the pin to the public key of the root CA. Alternatively, it is possible to associate the pin to the public key of any of the intermediate CAs existing in the current certification chain. A balance must be struck between the level of security (or restriction) to be established by HPKP, the possible accessibility risks, and the management tasks associated with the cryptographic keys. The considerations to be taken into account apply to both the main pin(s) and the backup pin(s).

The HPKP header must include at least two pins: a primary pin, linked to the SPKI value of one of the digital certificates in the current certification chain, and a backup pin, which must not be in the current certification chain

The most secure and restrictive scenario is the one that associates the pin with the public key of the web server itself. However, it must be carefully managed to modify the pins appropriately, anticipating the change and/or renewal of the key, and to include pins for all the keys associated to the web server (for example, if using RSA and ECDSA keys simultaneously; see section "3.2. Generation of cryptographic keys").

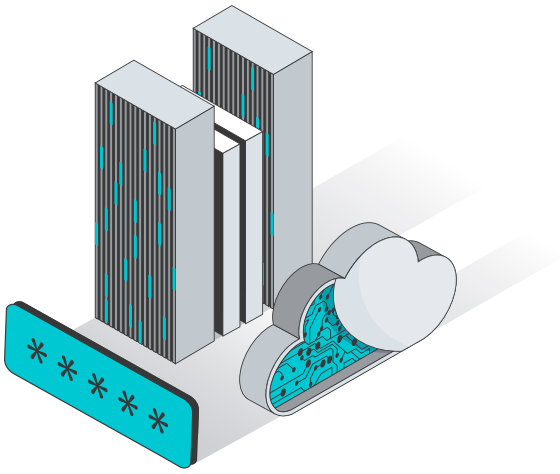
The most flexible and permissive scenario is the one that associates the pin to the public key of an intermediate CA, or even more so, of the root CA. However, it must also be carefully managed in order to maintain the current pins (if not to change them), especially during the renewal process of the web server's digital certificate, as CAs usually have several root and intermediate CAs. It is therefore important to ensure that the new digital certificate will be issued by the same CA. Also, CAs establish trust relationships with each other by cross-signing some digital certificates, which means that there can be several valid certificate chains for the same digital certificate. In this case it is necessary to have pins for each and every possible valid certification chain.

Trying to strike a balance between the two scenarios, and the advantages of each, one option is to create the pin for the first intermediate CA, i.e. the one that has directly issued the current digital certificate of the web server.

On the other hand, the backup pin (or pins) can be associated to the same CA, or even better, to a different CA, in case the current CA is compromised. It is also recommended to keep the private key and digital certificate associated to the backup pin offline, i.e. in a secure location where it is more difficult to be compromised.

Finally, it should be noted that HPKP does not offer protection against local manipulation of CAs in browsers and web clients, such as when a user imports a new root CA. The digital certificates associated to that root CA will be considered as valid, regardless of whether or not they match the pins of the HPKP header, a scenario that allows the use of local web proxies and enterprise solutions for inspection and interception of HTTPS traffic (see section "3.7.9. Inspection of HTTPS traffic").

The most secure and restrictive scenario is the one that associates the pin with the public key of the web server itself



3.3.10 Recommendations for the management of digital certificates

The summary of recommendations for the management of digital certificates, where multiple aspects need to be taken into account, includes:

- For publicly exposed web servers, it is necessary to have a digital certificate issued by a trusted and widely recognised public CA; self-signed digital certificates and private CAs are not valid.
- Evaluate and select the appropriate type of digital certificate (DV, OV or EV) to be obtained for each server or web application.
- Evaluate and select the CA that will issue the certificate using multiple objective criteria, such as its reputation, the functionality it offers, its adoption of new standards, the certificate revocation mechanisms it implements, etc.
- Select the digital certificate renewal period and renew certificates periodically and always before they expire.
- Make use of digital certificates signed using SHA-256 (or higher). Under no circumstances make use of signatures based on SHA-1 or MD5.
- Identify and properly configure (in the correct order) the complete certification (or certificate) chain associated with the digital certificate, from the root CA, through all intermediate CAs, to the final certificate.
- Select the appropriate type of digital certificate with respect to the name of the entity represented by it, i.e. direct or unique reference, list of names, wildcard certificate, or multi-domain.
- The digital certificate should include both the name of the represented web server ("www", or the list of web servers, or the wildcard) and its domain.
- The name of the entity must be reflected in the digital certificate through the "SAN" field (Subject Alternative Name, extension "subjectAltName") instead of the "CN" (Common Name).
- Ensure the validity of the digital certificate at all times, including entity name, validity period, certificate issuing CA, certificate revocation and purpose.

- Associate the digital certificate to a CA that makes use of Certificate Transparency (CT) and confirm that the certificate has been registered in the public CT logs.
- Monitor CT logs or records to identify the issuance of digital certificates of own domains illegitimately by other CAs, without the authorisation of the domain owner. Complementarily, make use of CT monitoring services to be informed of changes in the digital certificates associated with a domain.
- Provide CT registration information (via SCT) in HTTPS responses from the web server to web clients via an X.509v3 digital certificate extension, (or preferably) via a specific TLS extension or using OCSP Stapling.
- It is recommended to consider the use of the HTTP header "Expect-CT" to declare the need to use qualified digital certificates in TC. Failing this, make use of the notification capabilities of "Expect-CT" to identify the use of digital certificates that are invalid from a CT point of view.
- Configure in the DNS name resolution service the CAA records corresponding to the list of CAs associated to the domain's digital certificates.
- Associate the digital certificate to a CA that makes use of CAA (Certification Authority Authorisation).
- Make use of CAA notification capabilities to identify errors or invalid certificate requests.
- Do not use HPKP (HTTP Public Key Pinning) in any case.
- Select the CA linked to the HPKP backup pins and protect the private key associated with these pins accordingly.

3.4. SSL and TLS protocols

This section will go into all the variants and additional functionalities associated with the obsolete SSL (Secure Sockets Layer) protocol and its replacement, TLS (Transport Layer Security).

This section will go into all the variants and additional functionalities associated with the obsolete SSL (Secure Sockets Layer) protocol and its replacement, TLS (Transport Layer Security)

3.4.1 SSL and TLS protocol versions

The selection of the SSL and TLS protocols, and the specific versions of these protocols, to be used by the web server or application will be influenced by the security and interoperability requirements of the web environment, i.e. which web browsers and clients are to be supported and to allow access to the web environment with the highest possible level of security.

Different versions of the TLS protocol have introduced security enhancements. For example, TLS 1.0 added the use of HMAC-based Pseudo-Random Functions (PRF), with PRF also being used to generate the master secret. Padding capabilities were also improved in TLS 1.0 over SSL 3.0. TLS 1.1 mainly added security improvements over TLS 1.0, such as the use of CBC encryption with explicit initialisation vectors (IVs) in each TLS record, instead of predictable IVs, and again, padding improvements, and included TLS extensions (RFC 3546[54]). TLS 1.2 added authenticated encryption (AEAD), support for HMAC-SHA256 (this being used for the standard PRF function and thus for signing algorithms, although there is flexibility to use other PRF functions), an extension for both ends to indicate and negotiate which signing and hashing algorithms they support, and eliminated the use of weak cryptographic algorithms.

[54] https://tools.ietf.org/html/rfc3546

TLS1.3 adds further improvements, including the removal of weak ciphers and algorithms (only five are now recommended), simplified key exchange (and improved key exchange speed) and improved performance for already visited sites with mechanisms such as the "zero round trip time (0-RTT)" handshake.

The use of the SSL protocol is discouraged, as it is considered completely vulnerable from its version 2.0, due to previous vulnerabilities, but especially after the discovery of the DROWN vulnerability in March 2016, until its latest version 3.0, after the discovery of the POODLE vulnerability in October 2014 (see section "3.8. Vulnerabilities in HTTPS").

Therefore, in the most tolerant case from the point of view of interoperability, the web environment should only support the different versions of the TLS protocol. However, due to the existing weaknesses in version 1.0 of this protocol, such as (among others) BEAST, released in June 2011 (see section "3.8. Vulnerabilities in HTTPS"), it is recommended to strike a balance between security and interoperability and to support only TLS 1.1 and TLS 1.2 (RFC 5246[55]), along with future versions of the protocol, such as TLS 1.3 [Ref - 22][56].

For reference, all websites that accept credit card payments and are therefore regulated under PCI (Payment Card Industry), must remove support for TLS 1.0 by July 2018, a date that was initially set for July 2016[57] (denoting the obsolescence of this version of the protocol, which was published in January 1999[58]).

As of 25 March 2021, the date of the last update of RFC 8996, the TLS 1.0 and 1.1 protocols are obsolete.

TLS versions 1.2 and 1.3 are the only ones that support modern cryptographic algorithms, such as those associated with Authenticated Encryption (AE), or AEAD (see section "3.5.5. Strenght of algorithms and cryptographic suites").

Version 1.3 of the TLS protocol is already supported by all major browsers as of the following releases:

- **Google Chrome, version 67 and later**
- **Mozilla Firefox, version 61 and later**
- **Mac OS 10.3, iOS 11 and higher**

[55] https://tools.ietf.org/html/rfc5246

[56] https://datatracker.ietf.org/wg/tls/documents/

[57] https://blog.pcisecuritystandards.org/migrating-from-ssl-and-early-tls

[58] https://tools.ietf.org/html/rfc2246

3.4.2 SNI (Server Name Indication)

SNI (Server Name Indication) [Ref - 8] is an extension of the TLS protocol that allows web clients to indicate the name of the web server (hostname) they wish to connect to during the initial HTTPS connection establishment process using TLS (handshake).

TLS extensions are a mechanism available to extend the functionality of TLS without having to modify the protocol. Extensions have relevant positive implications from a security point of view, and are used for functionalities such as SNI, OCSP Stapling ("3.6.3. OCSP Stapling"), session tickets ("3.7.2.2. TLS session resumption or caching, and TLS session tickets"), or secure renegotiation mechanisms ("3.4.3. Renegotiation"), among others.

By using SNI it is possible to use HTTPS in environments with web servers or virtual hosts (HTTP/1.1), i.e. when different web servers (hostnames) coexist on the same physical server and use the same IP address. SNI therefore makes it possible to have different digital certificates for each of the web servers, all provided from the same IP address and HTTPS port (TCP/443).

If SNI support is not available, it is only possible to use one web server (or a very small group of servers) per IP address, as only one single digital certificate is associated with it. This digital certificate could be shared among several web servers associated to that IP address, but this option is only recommended if it is a digital certificate (e.g. wildcard) that is valid for all the different web servers of the same domain available at that IP address.

Since SNI is only not supported on older versions of Android <= 2.3.x, Java <= 6 and Internet Explorer on Windows XP (support for which ended in 2014), from an interoperability point of view, it is acceptable to use SNI.

SNI (Server Name Indication) [Ref - 8] is an extension of the TLS protocol that allows web clients to indicate the name of the web server (hostname) they wish to connect to during the initial HTTPS connection establishment process using TLS (handshake)

The only drawback of SNI is that the name of the web server is revealed in clear by the web client in network traffic before the encrypted tunnel is established via HTTPS, although in most scenarios this is not a drawback, as the name of the web server is publicly known or revealed via the DNS service.

In the case of multiple web servers or virtual hosts linked to the same IP address, it is recommended to have support for SNI in the HTTPS implementation.

3.4.3 Renegotiation

The TLS protocol implementation provides renegotiation mechanisms, which allow both the client and the server to renegotiate the details and parameters of an already established TLS session, e.g. when making use of client digital certificates, such as those associated with electronic ID cards (DNIe) or user certificates.

The TLS protocol implementation provides renegotiation mechanisms, which allow both the client and the server to renegotiate the details and parameters of an already established TLS session

The TLS renegotiation mechanisms were insecure, facilitating interception (MitM) and denial of service (DoS) attacks on the web server, as the renegotiation operation requires more resources on the server than on the client.

It is recommended to use a TLS implementation that makes use of the secure renegotiation mechanisms defined in RFC 5746[59] and, preferably, that the renegotiation is always initiated by the server. In case no use is made of client digital certificates, it is recommended to disable the renegotiation mechanisms completely. At the very least, mechanisms that allow renegotiation to be initiated by the client should be disabled. For example, Apache does not provide support for client-initiated renegotiation since version 2.2.15.

[59] https://tools.ietf.org/html/rfc5746

3.4.4 Downgrading of the TLS protocol version

One of the major weaknesses associated with HTTPS is the possibility that, during the negotiation process associated with the establishment of the encrypted session, a potential attacker can force the client and/or server to use a previous (downgrade) and vulnerable version of the SSL or TLS protocols. This technique was used, for example, in the POODLE attack to downgrade the connection to SSL version 3.0, even from more secure versions of the protocol, such as TLS 1.2.

To avoid such attack scenarios, it is recommended to implement in the web server TLS Fallback Signaling Cipher Suite Value (SCSV), RFC 7507[60], an extension to the TLS protocol that prevents protocol degradation attacks. This extension is supported from Chrome 33[61] and Firefox 35[62], and from OpenSSL version 1.0.1j.

3.4.5 HTTP/2

Currently, the biggest impact on the performance of TLS (see section "3.7.2. Performance") is usually not associated with cryptographically expensive operations from a CPU consumption point of view, but with network latency.

For this reason, and to improve the performance of the web environment, it is recommended to establish the minimum possible number of HTTPS (and therefore TCP) connections. This can be done by increasing the duration of TCP sessions (using the TCP keep alive parameter) or by using HTTP/2, a protocol that maintains a single TCP session on the web server for each web client. HTTP/2 is currently supported by most modern web browsers[63].

Additionally, the distance between the client and the web server is a critical factor in terms of communication latency. This distance can be optimised through the use of CDNs (Content Delivery Networks) with HTTPS capabilities, although it is recommended to evaluate the details of the HTTPS implementation of this type of services, and the different modalities offered by[64].

Currently, the biggest impact on the performance of TLS is usually not associated with cryptographically expensive operations from a CPU consumption point of view, but with network latency

60 https://tools.ietf.org/html/rfc7507

61 https://www.imperialviolet.org/2014/02/27/tlssymmetriccrypto.html

62 https://blog.mozilla.org/security/2014/10/14/the-poodle-attack-and-the-end-of-ssl-3-0/

63 http://caniuse.com/#feat=http2

64 https://www.troyhunt.com/cloudflare-ssl-and-unhealthy-security-absolutism/

3.4.6 SSL and TLS protocol recommendations

The summary of recommendations for SSL and TLS protocols includes:

- Completely disable support for SSL 2.0 and SSL 3.0.
- Completely disable support for the TLS 1.0 and TLS 1.1 protocol.
- Use only TLS protocol versions 1.1 and 1.2.
- Carry out the necessary actions to support TLS 1.3.
- Have support for SNI in the implementation of HTTPS in the case of multiple web servers or virtual hosts linked to the same IP address.
- Disable TLS renegotiation mechanisms, especially renegotiation that can be initiated by the client. If these capabilities are needed, enable server-initiated renegotiation and make use of secure TLS renegotiation mechanisms.
- Enable the TLS protocol extension "TLS Fallback Signaling Cipher Suite Value" (SCSV) to prevent TLS protocol downgrade attacks.
- Evaluate the possibility of implementing HTTP/2 on the web server, coexisting with the HTTP/1.1 version of the protocol, due to the performance advantages of using HTTPS.

3.5. Cryptographic algorithms and suites

The configuration of the algorithms and cryptographic suites supported by the HTTPS-based web environment is one of the most complex elements to take into account in the implementation of HTTPS, as it must always seek a balance between the level of security provided (based on the strength of the cryptographic algorithms and protocols used, and the industry developments and medium and long-term expectations of these), the performance of the web environment, and the interoperability with multiple browsers and web clients.

The configuration of the algorithms and cryptographic *suites* supported by the HTTPS-based web environment is one of the most complex elements to take into account in the implementation of HTTPS, as it must always seek a balance between the level of security provided, the performance of the web environment, and the interoperability with multiple browsers and web clients

3.5.1 Forward secrecy

RSA-based key exchange does not provide forward secrecy (also referred to as PFS, Perfect Forward Secrecy), i.e. a cryptographic property whose purpose is to prevent encrypted traffic captured in the past from being decrypted later if keys are compromised in the future. In other words, compromising or obtaining the current encryption keys does not allow the encryption keys used in the past to be compromised or obtained. Basically, forward secrecy prevents encrypted traffic from relying on the private key. Instead, Diffie-Hellman (DH) based algorithms should be used, such as DHE (DH Ephemeral, or EDH) or ECDHE (Elliptic Curve DHE, or EECDH), which make use of ephemeral DH parameters (and result in ephemeral keys). From a security and performance point of view, ECDHE is preferable to DHE.

Forward secrecy ensures that each TLS session from each client, even to the same server, makes use of different cryptographic keys, whereas without forward secrecy (e.g. when using RSA), the keys of all sessions depend on the web server's private key, and could be decrypted with it.

RSA, however, can still be used to generate the keys associated with digital certificates. It is important to differentiate between the cryptographic algorithms and keys used for the identification of the endpoints and the generation of the associated digital certificates (see section "3.2. Generation of cryptographic keys"), and the algorithms used for the exchange of keys during the establishment of a TLS session (see section "3.5.2. Robust Key Exchange"), where the use of forward secrecy must be ensured.

3.5.2 Robust Key Exchange

The paragraph "3.5.1. Forward secrecy" describes in detail the main property required during key exchange when establishing a TLS session: forward secrecy. For this purpose, cryptographic suites using RSA-based key exchange should not be used, but the elliptic curve variant (ECDHE) or, alternatively, the classical variant of Diffie-Hellman key exchange (DHE) using ephemeral keys (ephemeral). In reality, RSA is used to transport the master (or session) key, while (EC)DHE is used to negotiate the master key, using forward secrecy.

Three key exchange algorithms are therefore available (in reverse order of priority): RSA, DHE and ECDHE. In summary, key exchange via ECDHE should be performed with all modern browsers and web clients, and alternatively (for compatibility) DHE should be supported to make use of forward secrecy with older browsers and web clients (such as Android < 3.0, Java < 7 or OpenSSL < 1.0.0).

The nomenclature used for the key exchange algorithms (see section "3.5.5. Strenght of algorithms and cryptographic suites") includes the key exchange algorithm (RSA, DHE and ECDHE) and the authentication method (based on the type of keys used to generate digital certificates: RSA or ECDSA). The most common combinations, in reverse order of priority, are:

- **RSA:** RSA Key Exchange with RSA authentication.
- **DHE_RSA:** Ephemeral DH key exchange with RSA authentication.
- **ECDHE_RSA:** Ephemeral ECDH key exchange with RSA authentication.
- **ECDHE_ECDSA:** Ephemeral ECDH key exchange with ECDSA authentication.

NOTE:
Additionally, cryptographic suites that make use of post-quantum cryptography (CECPQ1) for key exchange[65], combining the NewHope algorithm with X25519 elliptic curves, have recently become available in TLS.

After the key exchange between the client and the server, regardless of the method used, both ends will have a premaster secret or premaster key. After applying the selected key exchange algorithm, and because each algorithm can handle different key lengths, the output value is processed by a Pseudo-Random Function (PRF) to always obtain a 48-byte (384 bits) master secret.

In TLS, key exchange is linked to the authentication process, with the public key (RSA or ECDSA) being used both to verify that communication is being established with the desired entity and to derive the master secret using the chosen key exchange algorithm. In the case of RSA, the client generates the master secret and sends it encrypted with the server's public key; only the server can decrypt it with the corresponding private key, implicitly authenticating itself. In the case of DHE or ECDHE, the DH parameters sent by the server are signed with its private key, so that the client can verify the signature with the corresponding public key, and confirm that it is communicating with the expected server (authenticating it).

[65] https://www.imperialviolet.org/2016/11/28/cecpq1.html

As a general recommendation, 256-bit key exchange algorithms should be used for ECDHE and 2.048-bit key exchange algorithms for DUSE (see section "3.5.3. Strength of Diffie-Hellman (DH) parameters").

It is therefore recommended to make use of ECDHE using, for example, the "secp256r1" curve (called "prime256v1" in OpenSSL and also known as P-256, and widely supported by web clients), which provides 128-bit security for key exchange. Alternatively, one could make use of the "secp384r1" curve (also known as P-384 and also generally supported, RFC 4492[66]), but which performs less well than P-256 and also does not provide a significant enough security improvement to justify its use. Alternatively, the "secp521r1" curve could be used, but its support is more limited.

These first two curves, defined by NIST, should be used with caution, as it is not clearly established how their parameters have been selected and therefore may have weaknesses known only to their designers. Alternatively, two standard curves known as Curve25519 (X2559) and Curve448 (X448) have recently become popular, including for use in TLS[67] [68].

Alternatively, for those cases where it is not possible for the web client to make use of ECDHE, it is recommended to provide support for DHE using robust 2.048 bit parameters (see section "3.5.3. Diffie-Hellman (DH) parameter strength").

.

As a general recommendation, 256-bit key exchange algorithms should be used for ECDHE and 2.048-bit

[66] https://tools.ietf.org/html/rfc4492

[67] https://tools.ietf.org/html/draft-ietf-tls-curve25519-01

[68] https://tools.ietf.org/html/draft-ietf-tls-rfc4492bis-15

3.5.3 Strength of Diffie-Hellman (DH) parameters

The length or size of the Diffie-Hellman (DH) parameters should be at least 2,048 bits and is recommended to match the length or strength of the corresponding private key.

Some known web client incompatibilities include Java 6, which only supports 1024-bit DH parameters. This is also the case for Java 7, but not conflicting from a security point of view, as it also supports ECDHE.

In any case, it is not recommended to use known or pre-configured DH groups (of 1024 bits), such as those recommended in RFC 3526[69], or DH parameters of 768 bits or less, especially after the discovery of the Logjam vulnerability in January 2015 (see section "3.8. Vulnerabilities in HTTPS"). Likewise, HTTPS implementations must be very careful in the selection of DH parameters, since, as the triple handshake vulnerability in 2014 demonstrated (see section "3.8. Vulnerabilities in HTTPS"), it was even possible for a malicious server to select insecure DH parameters that were not even prime numbers.

Until the introduction of RFC 7919[70] (August 2016), clients could not specify their security requirements with respect to DH parameters during the TLS session establishment and negotiation process, and had to accept those provided by the server. Based on RFC 7919 it is recommended, instead of using pre-configured DH groups or generating your own groups with "openssl dhparam", to make use of the DH groups pre-defined in that standard, as these have been audited from a security point of view: ffdhe2048, ffdhe3072 or ffdhe4096[71].

[69] https://tools.ietf.org/html/rfc3526

[70] https://tools.ietf.org/html/rfc7919

[71] https://wiki.mozilla.org/Security/Server_Side_TLS

3.5.4 Preference for cryptographic suites

One of the fundamental functionalities of the TLS protocol is the ability of the HTTPS client and server to negotiate, during the establishment of the TLS session (or TLS handshake), the set of cryptographic suites that could be used by both parties, and reach an agreement or consensus on which is the best option available depending on the priorities or preferences of both ends. This negotiation allows the web server or application to always try to make use of the most secure option available with each of the web clients that connect to it.

It is therefore recommended to configure the web server by first indicating the most robust cryptographic suites, listed in descending order of preference according to the level of security they offer.

Is recommended to configure the web server by first indicating the most robust cryptographic suites, listed in descending order of preference according to the level of security they offer

3.5.5 Strength of cryptographic algorithms and suites

The cryptographic suites used by TLS define how all communications between client and server will be carried out via HTTPS (using multiple cryptographic components or algorithms), including the authentication process to confirm that the communication is with the legitimate web server or application, through to all subsequent encrypted data exchange.

If any of the cryptographic algorithms used are weak or insecure, they should be replaced by a more modern one. The following cryptographic algorithms should not be used today:

- The cipher[72] (and authentication mechanism) NULL, as it does not provide encryption (and authentication) capabilities.
- Anonymous Diffie-Hellman (ADH) based suites as they do not provide authentication capabilities.
- Cryptographic suites with weak cipher algorithms based on keys with cryptologic strength below 100 bits.
- Cryptographic suites with cipher algorithms allowed for export (see FREAK attack in section "...") can be exported "3.8. Vulnerabilities in HTTPS").
- The RC4 encryption algorithm, as it is insecure
- The DES encryption algorithm, as it is insecure and slow, and its variant Triple DES (TDES or 3DES).
- MD5, RIPMED-160 and SHA-1 *hashing* algorithms, as they are insecure or weak.

TLS could be considered as a framework that allows different cryptographic protocols to be negotiated and used, selecting the components and parameters that will define the type of security to be used. The cryptographic suites used by TLS define in their name a set of cryptographic algorithms associated with the different components and phases of the TLS protocol. Among them, the key exchange method, the authentication method (or keys and certificates), the symmetric encryption algorithm and the length of the encryption keys to be used are specified (in the following order, as shown in the following figure [Ref - 10]), together with their mode of use (if multiple) and the hashing algorithm for the MAC (or the pseudo-random function algorithm, PRF, used for example in AEAD type algorithms, which do not make use of a MAC at the TLS level):

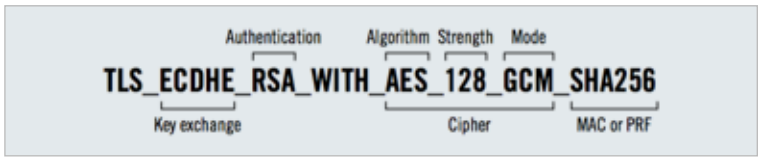


Figure 5.
Standard nomenclature (IANA) of the cryptographic suites used by TLS.
Source: [Ref - 10]

For example, the following cryptographic suites use the algorithms detailed below. All TLS protocol cryptographic suites begin with "TLS"

[72] The term 'cipher' is sometimes used to abbreviate 'encryption algorithm'.

Suite: TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
TLS with ECDHE key exchange, using ECDSA key (and digital certificate) authentication, with 128-bit AES encryption algorithm in GCM (Galois/Counter Mode) and using SHA-256 as a pseudo-random function (PRF).

Suite: TLS_DHE_RSA_WITH_AES_256_CBC_SHA
TLS with DHE key exchange, using RSA key (and digital certificate) authentication, with 256-bit AES encryption algorithm in CBC (Cipher Block Chaining) mode and using SHA-1 as hashing algorithm for MAC calculation.

The most complex part is the part associated with the final part of the name, i.e. the hashing algorithm for the MAC calculation, or the pseudo-random function (PRF) algorithm, since in TLS 1.2 and due to the flexibility that this version of the protocol provides, it can have different meanings. For example, for AEAD suites (such as AES/GCM and ChaCha20-Poly1305) the last part reflects the PRF function. For suites defined before TLS 1.2, the suite name defines the hashing algorithm for the MAC (calculation of the) in TLS (e.g. SHA-1), the PRF function is not reflected, as it is the protocol version (not reflected in the suite name) that defines it, since, for example, a suite may make use of HMAC-SHA256 as PRF for TLS 1.2, but HMAC-SHA1 as PRF for TLS 1.0 [Ref - 10].

It should be noted that OpenSSL uses a slightly different nomenclature to reference the cryptographic suites than the one defined in the IANA standard [Ref - 23] and used by other web technologies, such as IIS. Below are two examples of each nomenclature format (which, as can be seen, are quite similar, and reference the same code or unique identifier of each cryptographic suite):

Code: **0xC0,0x2B ó 0xC02B (RFC 5289)**
OpenSSL: **ECDHE-ECDSA-AES128-GCM-SHA256**
IANA: **TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256**

Code: **0x00,0x9E ó 0x009E (RFC 5288)**
OpenSSL: **DHE-RSA-AES128-GCM-SHA256**
IANA: **TLS_DHE_RSA_WITH_AES_128_GCM_SHA256**

NOTE:
A comparison of the IANA nomenclature with that used by other SSL/TLS libraries, such as GnuTLS, NSS and OpenSSL, is available[71].

It is therefore recommended to use cryptographic suites starting with "ECDHE-RSA-..." (preferably) or "DHE-RSA-...", in case of using RSA cryptographic keys, and "ECDHE-ECDSA-...", in case of using ECDSA cryptographic keys (see section "3.2. Generation of cryptographic keys"). The "ANNEX B. RECOMMENDED CRYPTOGRAPHIC SUITES" provides an initial reference list of preferred cryptographic suites for an initial HTTPS web server configuration (in OpenSSL format).

It should be noted that (in the future), when using the TLS 1.3 protocol, it does not negotiate the key exchange algorithm as part of the cryptographic suites, but the key exchange algorithm is negotiated separately (only algorithms that make use of forward secrecy, such as ECDHE and FFDHE, Finite Field Diffie-Hellman Ephemeral, RFC 7919, are supported in TLS 1.3)[70] (RFC 7919 70, August 2016). Therefore, the cryptographic suites supported by TLS 1.3[73] do not reflect in their name or definition the key exchange algorithm to be used. For example:

Code:	**0x13,0x01 ó 0x1301 (RFC5116)**
IANA:	**TLS_AES_128_GCM_SHA256**
Code:	**0x13,0x02 ó 0x1302 (RFC5116)**
IANA:	**TLS_AES_256_GCM_SHA384**
Code:	**0x13,0x03 ó 0x1303 (RFC7539)**
IANA:	**TLS_CHACHA20_POLY1305_SHA256**

TLS can make use of numerous encryption algorithms, both stream, block and authenticated encryption (AEAD) in TLS 1.2, such as 3DES, RC4, AES, CAMELLIA, ChaCha20, etc. Within the encryption algorithms, it is recommended to make use of those that provide Authenticated Encryption (AE), i.e. that natively combine encryption and integrity with data authentication, commonly known as AEAD (Authenticated Encryption with Associated Data). These modern algorithms also make use of strong keys and forward secrecy (see section "3.5.1. Forward secrecy"). For example, AES in GCM (Galois/Counter Mode) mode, as opposed to the more insecure CBC (Cipher Block Chaining) modes of AES, provides these features.

It should be noted that (in the future), when using the TLS 1.3 protocol, it does not negotiate the key exchange algorithm as part of the cryptographic suites, but the key exchange algorithm is negotiated separately

[73] https://tlswg.github.io/tls13-spec/#rfc.appendix.B.4

From a performance point of view, many CPUs today have support for operations that allow the AES algorithm to be run in hardware (AES-NI[74]), speeding up and optimising encryption and decryption operations.

The validation of the integrity of the data exchanged in TLS, through a MAC (Message Authentication Code), is implicit in the encryption process.

After obtaining the master secret, (together with two random seeds from the client and the server) it is processed by a Pseudo-Random Function (PRF) to obtain the necessary cryptographic material depending on the cryptographic suite to be used, and mainly on the encryption algorithms. Since each encryption algorithm requires cryptographic material of a different length, the output value will be adapted according to the suite negotiated.

It is therefore recommended to use AEAD encryption using AES 128-bit or 256-bit GCM encryption for TLS (1.2), as defined in RFC 5288[75], dated August 2008. Additionally, and once standardised for use in TLS, the ChaCha20 stream cipher can be used, together with the Poly1305 authenticator, as defined in RFC 7905[76], dated June 2016.

From the point of view of browsers and web clients, it is possible to evaluate their support for robust or weak and vulnerable algorithms and cryptographic suites through the "SSL Client Test" service of Qualys SSL Labs [Ref - 3], and from the point of view of a bad implementation, through the "BadSSL" service [Ref - 20] and, in particular, its "Dashboard"[77]. The "BadSSL" service allows checking the behaviour of web clients against incorrect or undesired HTTPS configurations. The associated domain, "badssl.com", provides a multitude of different servers (and hostnames), each of them with specific weaknesses and vulnerabilities, making it possible to check the HTTPS connection capabilities of a web client in this type of vulnerable scenarios, as well as to obtain (from the user's point of view) the specific error message, alert or warning generated in each undesired situation.

The validation of the integrity of the data exchanged in TLS, through a MAC (Message Authentication Code), is implicit in the encryption process

74 AES-NI: (Intel) Advanced Encryption Standard New Instructions

75 "AES Galois Counter Mode (GCM) Cipher Suites for TLS": https://tools.ietf.org/html/rfc5288

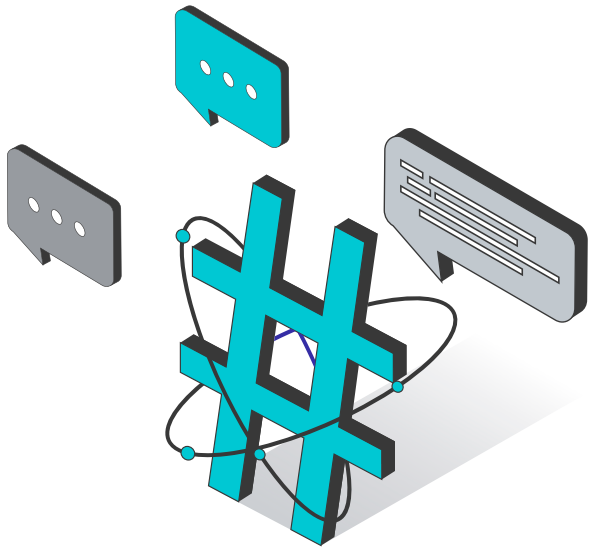
76 "ChaCha20-Poly1305 Cipher Suites for TLS": https://tools.ietf.org/html/rfc7905

77 https://badssl.com/dashboard/

In summary, once again, a balance between security and performance must always be sought, so currently for standard web environments, for the generation of cryptographic keys it is recommended to use 256-bit ECDSA keys and 2,048-bit RSA keys (see section "3.2. Generation of cryptographic keys"), for key exchange it is recommended to use 256-bit ECDHE and 2,048-bit DHE keys (see section "3.5.2. Robust Key Exchange") and, as a general recommendation, symmetric encryption algorithms providing at least 128-bit security should be used (the performance impact of 256-bit symmetric encryption algorithms should be assessed with respect to the increase in security they provide). Also, as a general recommendation, hashing algorithms based on SHA-256 should be used, and more robust algorithms (e.g. SHA-384, SHA-512...) should be used in very specific cases, which require a higher degree of security.

It should be noted that when using cryptographic suites that use the SHA-1 hashing algorithm for the MAC (reflected at the end of its name simply as "SHA"), they actually use it with a key in its variant HMAC (keyed-Hash Message Authentication Code), i.e. HMAC-SHA1, for which there are no known attacks and which can therefore be used with security guarantees (unlike the use of SHA-1 without HMAC).

As a general recommendation, hashing algorithms based on SHA-256 should be used, and more robust algorithms should be used in very specific cases, which require a higher degree of security



3.5.6 Recommendations for algorithms and cryptographic suites

The summary of recommendations for algorithms and cryptographic suites includes:

- Use key exchange algorithms that provide forward secrecy, preferably ECDHE, or alternatively DHE, as opposed to RSA.
- 256-bit key exchange algorithms should be used in the case of ECDHE and 2,048-bit key exchange algorithms in the case of DHE.
- If ECDHE is used, the type of elliptic curves to be used must be evaluated in detail. A couple of widely supported curves are available, such as secp256r1(P-256) and secp384r1 (P-384), defined by NIST and which have not been exhaustively analysed publicly, and new curves considered robust, such as Curve25519 (X2559) and Curve448 (X448), whose support is starting to become popular.
- In the case of using DHE, the size of the DH parameters should be at least 2,048 bits, and it is recommended to match the length or strength of the corresponding private key.
- When using DHE, it is not recommended to use known or pre-configured DH groups, such as those in RFC 3526, or to generate your own groups. Instead, it is preferable to make use of the pre-defined DH groups in RFC 7919: ffdhe2048, ffdhe3072 or ffdhe4096.
- Configure the web server giving preference to the most robust cryptographic suites, listed in descending order of preference according to the level of security they offer.
- Do not use the NULL cipher, ADH-based suites, cipher algorithms with keys with cryptologic strength below 100 bits, suites with cipher algorithms allowed for export, RC4, DES, 3DES (or TDES) ciphers, MD5, RIPMED-160 and SHA-1 hashing algorithms, etc.
- When prioritising TLS cryptographic suites, with respect to the key exchange and authentication algorithm, it is recommended to use cryptographic suites starting with "ECDHE-ECDSA-...", in case of using ECDSA cryptographic keys, and with "ECDHE-RSA-..." (preferably) or "DHE-RSA-...", in case of using RSA cryptographic keys.

- Make use of encryption algorithms that provide Authenticated Encryption (AE) in TLS 1.2, commonly known as AEAD (Authenticated Encryption with Associated Data), such as AES 128-bit or 256-bit GCM (Galois/Counter Mode), as opposed to AES CBC (Cipher Block Chaining) modes, or the ChaCha20 stream cipher, together with the Poly1305 authenticator.

- Symmetric encryption algorithms providing at least 128-bit security should be used (the performance impact of 256-bit symmetric encryption algorithms should be assessed against the increase in security they provide).

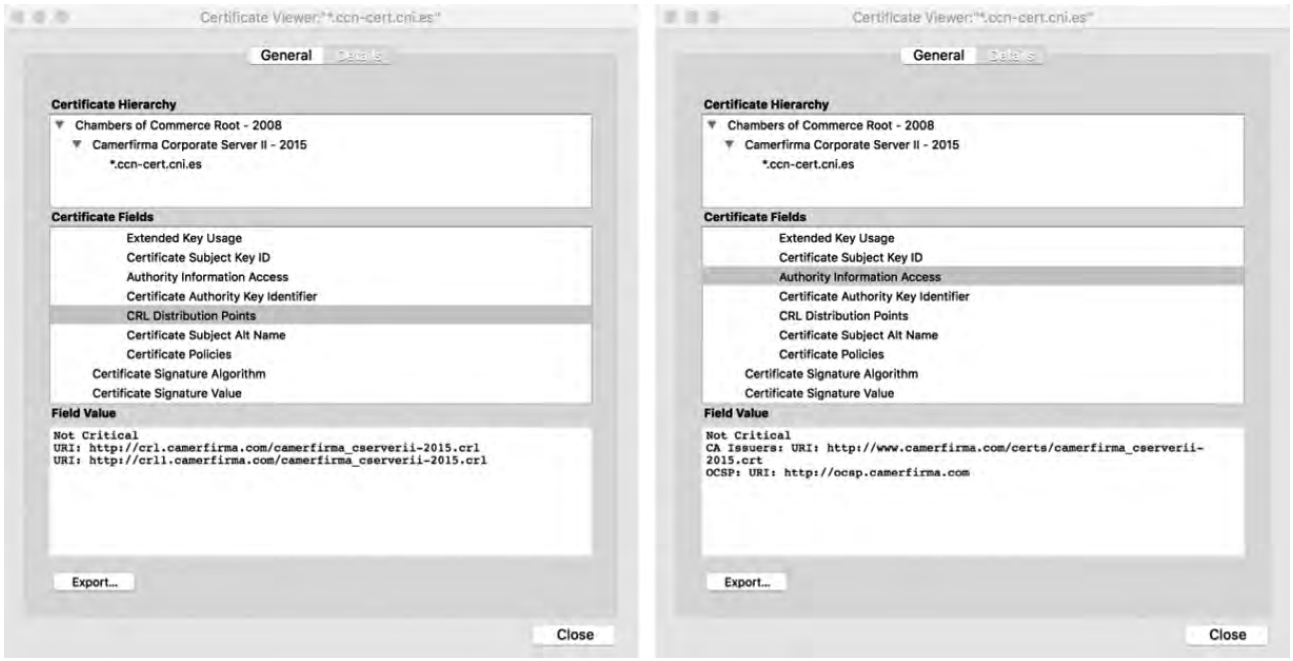
- Hashing algorithms based on SHA-256 should be used, and more robust algorithms (e.g. SHA-384, SHA-512...) should be used in very specific cases, which require a higher degree of security.

3.6. Digital certificate revocation mechanisms

Digital certificate revocation mechanisms allow the certificate owner to indicate to users (browsers and web clients) that the certificate is no longer valid, as it has probably been replaced by a new one.

Digital certificates must include revocation mechanisms such as CRLs (Certificate Revocation Lists) or OCSP (Online Certificate Status Protocol) or, preferably, OCSP Stapling should be used. The digital certificate itself provides instructions or references for using the first two mechanisms to verify the validity of the certificate and whether or not it has been revoked:

Figure 6.
Revocation information included in digital certificates.



3.6.1 Certificate Revocation Lists (CRLs)

The CRL (Certificate Revocation List) mechanism, RFC 6818[78], allows browsers and web clients to verify the validity of the status of a digital certificate by downloading the revocation list (hereinafter CRL list), which is maintained by the CA that issued the certificate. If the certificate's serial number appears on the list, it means that it has been revoked.

The reference to the CRL list can be found in the "CRL Distribution Points" field of the digital certificate.

The main limitations of CRL are its scalability, due to the size of the revocation lists, which increase with each new certificate revoked, the absence of recent or updated information (since due to the size of the lists, web clients usually keep a local copy in cache), the performance implications when completing the TLS handshake and verifying the certificate (considering that it is necessary to download the CRL list at connection time), and the handling of failures in downloading the CRL list.

If for any reason it is not possible to download the CRL list, it is usual for web clients to make use of the soft fail mode and consider the certificate as valid.

The use of CRLs by browsers and web clients is nowadays tending towards disuse, or to very specific usage scenarios (e.g. EV certificates) or not widespread, almost always used as a last resort for checking.

The CRL allows browsers and web clients to verify the validity of the status of a digital certificate by downloading the revocation list, which is maintained by the CA that issued the certificate

[78] https://tools.ietf.org/html/rfc6818

3.6.2 Online Certificate Status Protocol (OCSP)

The OCSP (Online Certificate Status Protocol) mechanism, RFC 6960[79], allows browsers and web clients to verify the validity of the status of a digital certificate by making a real-time query to the OCSP service (known as OCSP response service or OCSP responder), which is maintained by the CA that issued the certificate. The query to the OCSP service is carried out only for the certificate's serial number, optimising the bandwidth needed to check the revocation status, and the response obtained allows knowing whether it has been revoked or not.

The reference to the OCSP service is in the "(Certificate) Authority Information Access" field (in the "OCSP" line) of the digital certificate.

The main limitations of OCSP are that the CA must have a fast, reliable and up-to-date OCSP service that is always available and can handle high peak workloads, and again, the performance implications when completing the TLS handshake and verifying the certificate (considering that it is necessary to receive the response from the OCSP service at the time of connection), together with the management of access failures to the OCSP service. On the other hand, and from a privacy point of view, the CA can know which websites are being visited by a web client.

If for any reason it is not possible to access the OCSP service, it is common for web clients to make use of the soft fail mode and consider the certificate as valid. It is relatively easy to carry out attacks (via TCP errors, HTTP errors or even errors in the OCSP protocol itself) that prevent the web client from receiving the response from the OCSP service.

OCSP responses can be valid for up to 10 days for final digital certificates (typically 7 days), and up to 12 months for digital certificates of intermediate CAs. Additionally, certain performance optimisations applied on OCSP facilitate replay attacks.

The OCSP mechanism, RFC 6960, allows browsers and web clients to verify the validity of the status of a digital certificate by making a real-time query to the OCSP service, which is maintained by the CA that issued the certificate

[79] https://tools.ietf.org/html/rfc6960

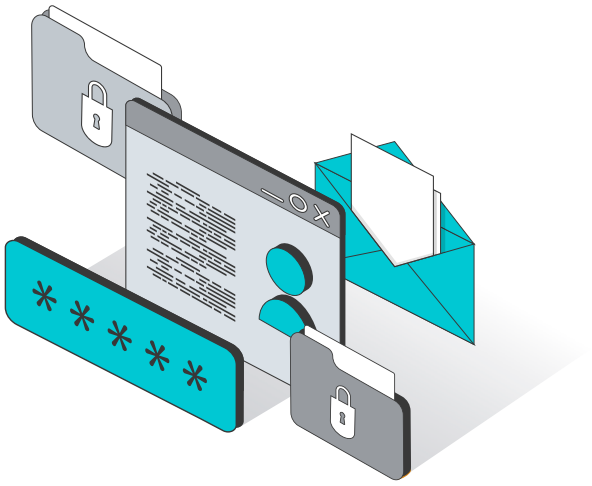
When selecting a CA, it is recommended to verify that the CA used for issuing digital certificates (see section "3.3.2. Issuing digital certificates: Certification Authorities (CAs)") has a fast and reliable OCSP service.

Again, as with CRLs, the use of OCSP by browsers and web clients today is very varied, and it is not usually active by default, or only in very specific usage scenarios (e.g. EV certificates) or not generalised, so it cannot be considered effective, as demonstrated by the appearance of other proprietary alternatives such as CRLSets in Chrome or OneCRL in Firefox.

If a web client makes use of OCSP, each time it connects to the web server, it will also have to connect to the CA to verify the current validity of the digital certificate. OCSP stapling (see section "3.6.3. OCSP Stapling") allows to avoid this continuous dependency on the CA (with the associated positive implications of availability, performance and privacy).

.

If a web client makes use of OCSP, each time it connects to the web server, it will also have to connect to the CA to verify the current validity of the digital certificate



3.6.3 OCSP Stapling and OCSP Must-Staple

If a web client supports and requests OCSP Stapling, and the web server also implements OCSP Stapling, each time the client connects to the web server, the web server can directly provide the OCSP response so that the client can verify the current validity of the digital certificate, thus avoiding most of the negative implications associated with the use of OCSP or CRL.

OCSP Stapling[80], RFC 6066[81], was designed because neither CRL nor OCSP provide the desired security and performance for certificate revocation checking on HTTPS connections. In most cases, revocation checks have not been effective, as was demonstrated in 2011 when it was necessary to distribute updates to browsers and web clients in order to add to the revocation blacklists the fake certificates generated after the compromise of several CAs. With OCSP Stapling, instead of the web client having to check the revocation status, the web server does it periodically and provides the details to the client.

The TLS extension associated with OCSP Stapling allows the web server to provide the most up-to-date information possible on the revocation status of a digital certificate to the web client, sent (attached or stapled, hence its name) during the establishment of the TLS connection or handshake, from the web server itself (without depending on the CA at the time of the connection). To do this, the server periodically obtains the status of the certificate by means of an OCSP request to the CA service, receiving a response signed by the CA and with a timestamp. When the web client receives such a response during the establishment of the TLS connection, it can verify the signature of the OCSP response and thus the validity of the certificate. The client can also trust the response as it has a limited validity in time (based on the timestamp).

If for any reason it is not possible to obtain the OCSP Stapling response, and the web server declares that it makes use of this revocation mechanism (via the OCSP Must-Staple directive, described below), web clients may make use of the hard fail mode and not consider the certificate as valid.

If a web client supports and requests OCSP Stapling, and the web server also implements OCSP Stapling, each time the client connects to the web server, the web server can directly provide the OCSP response so that the client can verify the current validity of the digital certificate

[80] https://scotthelme.co.uk/ocsp-stapling-speeding-up-ssl/

[81] https://tools.ietf.org/html/rfc6066

In the initial usage scenario of OCSP Stapling, the web client cannot know whether the website supports OCSP Stapling and therefore should expect to receive an OCSP response in the TLS handshake. To solve this scenario, OCSP Must-Staple was created.

OCSP Must-Staple[82], RFC 7633[83], defines a new setting associated with X.509 digital certificates that must be activated by the CA, indicating to the browser or web client that the digital certificate must be offered through a TLS session that includes an OCSP Stapling response. Otherwise, the web client must consider that the digital certificate is not valid (hard fail).

The process used to generate a certificate signing request (CSR) to the CA must include the OCSP Must-Staple extension.

Because the combination of OCSP Stapling together with OCSP Must-Staple is more restrictive than the previously available revocation options, and because scenarios could arise that affect the availability of the web environment if not implemented correctly (as with HPKP or CSP), it is recommended to implement it with caution and to evaluate its implementation in detail. To this end, a new notification mechanism, called OCSP Expect-Staple[84], is available that allows a website owner to monitor the operation of OCSP Stapling prior to its strict activation via OCSP Must-Staple. Its monitoring capabilities are similar to those associated with the "report-uri" directive of security HTTP headers such as HPKP (see section "3.3.9. Restriction of legitimate digital certificates: HPKP") or CSP (see section "3.7.6. Content Security Policy (CSP)").

To this end, a new notification mechanism, called OCSP Expect-Staple, is available that allows a website owner to monitor the operation of OCSP Stapling prior to its strict activation via OCSP Must-Staple

The preloaded list of HSTS (see section "3.7.4. Forcing strict (default) use of HTTPS: HSTS") is becoming the reference location in web browsers to reflect the security requirements of web environments. This list contains not only the HSTS directive itself, but also the HPKP prefixed pins (for certain domains only), Expect-CT (see "3.3.6. Certificate Transparency (CT)") and OCSP Expect-Staple. At the time of writing, the process of adding OCSP Expect-Staple to the web environment is done manually with the Chromium/Chrome preloaded list maintainers.

82 https://scotthelme.co.uk/ocsp-must-staple/

83 https://tools.ietf.org/html/rfc7633

84 https://scotthelme.co.uk/ocsp-expect-staple/

The definition of OCSP Expect-Staple in that list includes the statement that responses from the web environment are expected to make use of OCSP Stapling and include an OCSP response, the URL to which to submit error reports associated with OCSP Expect-Staple, and whether the policy applies equally to all subdomains (includeSubDomains).

When the OCSP Expect-Staple policy is active, if the web browser establishes a connection to the web environment but does not receive a valid OCSP response via OCSP Stapling, it will generate a report notifying this situation to the previously defined URL, using the format described in the specification[85]. The report not only notifies if the OCSP response has not been received (absent), but also if it has been received, but is invalid for other reasons (expired, not applicable for the received digital certificate, incorrect format, etc.). These details provide the necessary information to configure and implement an optimal OCSP Stapling environment.

Initially, OCSP Stapling could only provide information about the web server's digital certificate, but RFC 6961[86] (June 2013) extends its capabilities to provide revocation details of multiple certificates simultaneously, such as those associated with intermediate CAs reflected in the certification chain.

The advantage of using OCSP Stapling together with OCSP Must-Staple is that the time in which web clients are notified of a potential compromise of a digital certificate is significantly reduced from a maximum of 39 months (maximum validity period of digital certificates, or 825 days with the new proposal as of March 2018, see section "3.3.5. Renewal of digital certificates"), to the maximum associated with the lifetime of an OCSP response, i.e. normally a maximum of 7 days.

The advantage of using OCSP Stapling together with OCSP Must-Staple is that the time in which web clients are notified of a potential compromise of a digital certificate is significantly reduced from a maximum of 39 months, to the maximum associated with the lifetime of an OCSP response

85 https://docs.google.com/document/d/1aISglJIIwglcOAhqNfK-2vtQl-_dWAapc-VLDh-9-BE/edit#heading=h.rkpittae54q

86 https://tools.ietf.org/html/rfc6961

3.6.4 Recommendations for revocation of digital certificates

The summary of recommendations for the revocation of digital certificates includes:

- Digital certificates should include revocation mechanisms such as CRLs (Certificate Revocation Lists) or OCSP (Online Certificate Status Protocol) or, preferably, OCSP Stapling should be used.
- When selecting a CA, it is recommended to verify that it has a fast and reliable OCSP service.
- Make use of the TLS OCSP Stapling extension to provide clients with up-to-date information on the revocation status of a digital certificate.
- Make use of a CA that allows adding the OCSP Must-Staple policy to digital certificates, to allow web clients to consider the certificate as invalid if it is not possible to obtain a valid OCSP Stapling response.
- Make use of OCSP Must-Staple. Complementarily, make use of the OCSP Expect-Staple notification capabilities, which allow monitoring the operation of OCSP Stapling before its strict activation by OCSP Must-Staple. These capabilities, still available on an experimental basis, allow for a progressive and controlled use of OCSP Must-Staple.
- Provide revocation details of multiple certificates simultaneously, such as those associated with intermediate CAs, by OCSP Stapling (if deemed necessary).

3.7. HTTPS web content and applications

The following sections provide numerous recommendations affecting the design and configuration of both the web server and its contents, as well as web applications, all focused on providing maximum coverage to offer all available content in the web environment via HTTPS.

3.7.1 Priority in web search engines

One of the objectives of most websites is to be well positioned in Internet search engines (such as Google, Yahoo!, Bing, etc.) in order to be easily found and identified by potential users.

In August 2014, Google announced [Ref - 24] that it would give more relevance in the positioning (or ranking) of searches, associated with SEO (Search Engine Optimisation) algorithms, to websites that made use of HTTPS as opposed to those that used HTTP. Subsequently, in December 2015 Google announced that the search and indexing of web pages would be carried out by default for the HTTPS version, i.e. starting with HTTPS pages as opposed to HTTP[87], even if no references to HTTPS have been identified.

For this reason, in order to facilitate the indexing of the contents of the website, it is recommended not to add the "noindex" directive for URLs that make use of HTTPS, neither through the HTML tag "<meta>", nor through the HTTP headers.

.

One of the objectives of most websites is to be well positioned in Internet search engines in order to be easily found and identified by potential users

[87] https://security.googleblog.com/2015/12/indexing-https-pages-by-default.html

In addition, the policy established in the "robots.txt" file must allow the indexing of the HTTPS contents of the web server by web search engines.

Also related to web search engines and SEO algorithms, in relation to the existence of duplicate content and the redirection of any resource from HTTP to HTTPS (see section "3.7.3. Forcing the use (by default) of HTTPS"), it is recommended to make use of the "rel=canonical" tag[88] [89], which allows to define which is the canonical or reference resource within the web server for similar (or duplicated) contents.

The "rel=canonical" tag must be added to the "<head>" header of all duplicate web pages (and even unique or non-duplicate web pages), specifying the referring HTTPS web page for web search engines:

```
<link rel="canonical" href="https://www.dominio.es/p.php?i=1" />
```

In case the global redirection from HTTP to HTTPS does not apply to all resources of the web application, this tag allows adding an additional indication of the intention to use HTTPS, for indexing by web search engines.

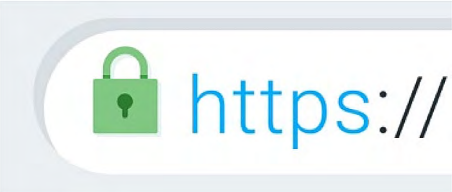


Figure 7.
Priority of HTTPS websites in web search engines.
Source: Google

[88] https://webmasters.googleblog.com/2009/02/specify-your-canonical.html

[89] https://webmasters.googleblog.com/2013/04/5-common-mistakes-with-relcanonical.html

3.7.2 Performance

In the past, one of the arguments against the use of HTTPS over HTTP has been the impact on server or web application performance. Numerous improvements and optimisations in the protocols associated with HTTPS, as well as in the implementations [Ref - 9], have significantly increased the performance of HTTPS and the loading time of web pages that use this protocol, and even allowed it to exceed the performance of HTTP, especially if HTTP/2 is used (see section "3.4.5. HTTP/2").

Among the performance improvements and optimisations (applied to CPU usage, connection latency, etc.) available are TLS handshake (1-RTT), TLS record size and certification chain optimisations, TLS session resumption or TLS session caching (TLS session tickets), TLS False Start, TCP Fast Open, TLS early termination, OCSP Stapling, etc., along with new features that will be incorporated in TLS 1.3, such as 0-RTT or zero-RTT.

Numerous resources are available[90] to evaluate the performance of HTTP vs HTTPS, such as "HTTP vs HTTPS Test"[91].

Due to attacks such as CRIME (see section "3.8. Vulnerabilities in HTTPS"), it is recommended to disable the TLS compression capabilities of the web server. However, the possibility of using Brotli compression mechanisms is available [Ref - 25], but only for HTTPS connections, where a web client can indicate that it has support through the HTTP header "Accept-encoding: br". These capabilities were included in Firefox 44[92], Chrome 55[93], Edge 15 or Opera 38[94]. Brotli provides improvements of around 20-26% over other existing compression algorithms, reducing the bandwidth used for web communications, and speeding up the loading time of web pages and resources. It is recommended to check if Brotli compression support is available on the web server, and enable it if available.

Numerous improvements and optimisations in the protocols associated with HTTPS, as well as in the implementations [Ref - 9], have significantly increased the performance of HTTPS and the loading time of web pages that use this protocol

90 https://scotthelme.co.uk/still-think-you-dont-need-https/

91 https://www.httpvshttps.com

92 https://www.mozilla.org/en-US/firefox/44.0/releasenotes/

93 https://groups.google.com/a/chromium.org/forum/#!searchin/blink-dev/brotli/blink-dev/JufzX024oy0/WEOGbN43AwAJ

94 http://caniuse.com/#feat=brotli

3.7.2.1 TLS FALSE START

The establishment of a TLS session, via a standard TLS handshake, requires the use of 2-RTT (roundtrips) between the client and the server [Ref - 9], affecting the performance of HTTPS connections.

TLS False Start is an extension to the TLS protocol that allows the transmission of encrypted application data when the TLS session is only partially established, thus reducing the TLS session establishment to 1-RTT. TLS False Start optimises TLS performance by slightly sacrificing TLS security, as data is sent before the integrity of the TLS handshake is verified.

TLS False Start is supported by all modern web browsers and clients, but to be used with confidence and to counteract the security sacrifice mentioned above, they verify that the web server makes use of (at least) 128-bit cipher suites, with forward secrecy, and the TLS NPN extension[95] (Next Protocol Negotiation, previously) or ALPN (Application Layer Protocol Negotiation).

If you combine TLS False Start together with TLS session resumption (or TLS session caching, see section "3.7.2.2. TLS session resumption or caching, and TLS session tickets") it is possible to make use of 1-RTT TLS handshakes for new sessions, and optimise cryptographic calculations for subsequent (resumed) sessions.

In addition, and in order to be able to compare TLS protocol versions with each other, one of the design goals of TLS 1.3 (see section "3.4.1. SSL and TLS protocol versions") is to have capabilities to be able to establish new TLS sessions in 1-RTT and summarise sessions in 0-RTT.

3.7.2.2 TLS SESSION RESUMPTION OR CACHING, AND TLS SESSION TICKETS

If a client has previously communicated with the server via TLS, it is possible to establish a new session in an abbreviated manner, and reduce CPU usage by reusing cryptographic parameters from the previous session.

TLS session resumption, also referred to as TLS session caching (RFC 5077[96]), is a mechanism that optimises the performance of TLS by slightly sacrificing its security. When establishing a TLS session, the client and server negotiate and derive a master key, a cryptographically expensive operation. If TLS session caching is used, subsequent connections that are part of the same session will make use of the

[95] https://www.imperialviolet.org/2012/04/11/falsestart.html

[96] https://tools.ietf.org/html/rfc5077

same master key, reducing the connection time (to 1-RTT). If a potential attacker obtains the master key, he could decrypt all connections associated with the same session. However, since TLS sessions should not last over time, their use is acceptable from a security point of view, considering the performance benefits obtained.

It is recommended to set the maximum period in which TLS sessions will be cached, using for example one day (24 hours) as a reference. It is also recommended not to share the TLS session cache between different environments, servers or web applications (in shared hosting web environments).

Sessions are identified both on the client and on the server by a session identifier (Session ID - RFC 5246[55]), exchanged between both parties during the initial establishment of the session. When using TLS session caching, both the client and the server must maintain session state information for a period of time.

The alternative (to avoid server-side storage, especially when the server handles thousands or millions of web clients) is to make use of session tickets (TLS session tickets - RFC 507796), where all session state is kept on the client (also known as stateless TLS session resumption). From a security point of view, the use of TLS session tickets is discouraged, as they expose all cryptographic details of the session state over the communication channel, protected only with a ticket key, which, in some cases, may be weaker than the session keys themselves. For example, OpenSSL uses AES 128 bits for ticket keys and, by default, reuses ticket keys between sessions, which are created at web server start-up and are not rotated (they can be used for very long periods of time and effectively override forward secrecy). Also, in the case of using TLS session tickets, it is recommended not to share the ticket key between different environments, servers or web applications (in shared hosting web environments).

For example, Apache 2.4.12+ has advanced capabilities for TLS session cache management, the ability to disable session tickets and ticket key management.

3.7.2.3 PRECONNECT

The HTML <link> tag[97] specifies relationships between the current web document and other external resources, such as style sheets (CSS or stylesheets), but can also be used to reference other (e.g. external) web resources.

The HTML "preconnect" directive[98] [Ref - 26] of <link> (<link rel="preconnect" href="https://...">) allows you to tell the browser or web client to prematurely open an HTTPS connection to the specified server and port to load resources that will be referenced later. These pre-indications improve performance by performing DNS resolution, TCP connection establishment and TLS negotiation in advance. The "preconnect" directive is supported in Chrome 46[99], Firefox 39[100], and Opera 33[101], and complements other performance-enhancing web directives such as "preload", "prefetch" or "prerender"[102].

3.7.3 Forcing the use (by default) of HTTPS

One of the problems still underlying web technologies is that HTTP is still used by default, and not HTTPS, which is an optional feature. In order to implement a web server or application only available over HTTPS, it is recommended to automatically redirect all HTTP traffic (TCP/80) to HTTPS (TCP/443; see section "3.1. Access to TCP/IP ports associated with web services").

Under no circumstances should a user accessing the HTTPS version of the web environment, server or application be redirected to the HTTP version, which does not make use of authentication and/or encryption mechanisms.

97 https://developer.mozilla.org/en-US/docs/Web/HTML/Element/link

98 https://www.igvita.com/2015/08/17/eliminating-roundtrips-with-preconnect/

99 https://www.chromestatus.com/feature/5560623895150592

100 https://bugzilla.mozilla.org/show_bug.cgi?id=1135160

101 http://caniuse.com/#search=preconnect

102 https://www.keycdn.com/blog/resource-hints/

3.7.4 Forcing strict (default) use of HTTPS: HSTS

The HTTP protocol security header known as HSTS or STS (HTTP Strict Transport Security) [Ref - 27] allows the web server to declare (and inform web clients) that it is only accessible via HTTPS (and not HTTP). As a result, browsers and web clients with HSTS support will not generate any HTTP requests to the web environment, automatically employing HTTPS at all times.

HSTS prevents configuration and implementation errors, and both passive MitM attacks, based on capturing unencrypted traffic, and active attacks, known as SSLstrip, i.e. where a potential attacker forces the victim web client to connect to the web server using HTTP, without using encryption, instead of HTTPS and, as a result, revealing sensitive information such as credentials, cookies or session IDs, etc.

Additionally, the use of HSTS mitigates attacks due to the possibility of the user accepting invalid digital certificates. Security alerts and errors in the validation of digital certificates in web browsers cannot be ignored or avoided by users when using HSTS.

Security alerts and errors in the validation of digital certificates in web browsers cannot be ignored or avoided by users when using HSTS

The HSTS header must be included in each and every HTTPS response generated by the web server or application:

```
Strict-Transport-Security: max-age=31536000 [; includeSubDomains]
[; preload]
```

It should be noted that the web browser enforces the use of HTTPS for a particular web server for the period indicated by the "max-age" directive (specified in seconds) in the HSTS header.

It is recommended that the "max-age" value be at least 10886400 (18 weeks), as this is the minimum value required to be admitted to the preloaded list (described below). However, it would be advisable to make use of longer time periods of at least 6 or 12 months (or 365 days: "31536000").

To minimise the possible impact of an incorrect configuration when starting the HSTS deployment, it is recommended to progressively increase the "max-age" value, starting with a small value (5 minutes: "300"), and increasing it to larger values (1 week: "604800", 1 month: "2592000", and finally to its final value). At the same time, the web server's access logs should be thoroughly monitored for requests received over HTTP (and not HTTPS).

From a security point of view, HSTS must be implemented for the entire domain, and not just for individual web servers. For this purpose, the "includeSubDomains" directive must be used. To avoid accessibility problems, all web servers and web applications in the domain must use HTTPS and have valid digital certificates. The application of HSTS to all subdomains mitigates attacks based on the manipulation of the DNS service and the generation of new names of web servers that do not exist in reality, and on which HTTPS connections would not be used (as they do not exist, there is no HSTS header, or a specific directive in the preloaded list, for them in the web clients).

From a security point of view, HSTS must be implemented for the entire domain, and not just for individual web servers

Additionally, it is important to note that HSTS is a TOFU (Trust On First Use) security mechanism, i.e. it does not apply the first time the web client connects to the web server. To mitigate this weakness, the "preload" directive indicates the interest and the possibility that the web server or the associated domain has been included in the HSTS preload list of web browsers, "HSTS Preload List". In reality, the preloaded list only covers entire domains, and it is not possible to add individual web servers.

The HSTS preloaded list [Ref - 28] is an internal list in web browsers, managed by Chrome/Google, which is used as a reference by all other web browsers to determine which domains want to make exclusive use of HTTPS and therefore for which the web browser will never generate HTTP traffic (thus avoiding attacks even on the first connection of the web client).

In order for a domain to be admitted to the preloaded list, it must meet the minimum requirements [Ref - 28][103] (as of February 2016), similar to those described above, and maintain them from the moment it is included in the list. It is possible to check the complete preloaded list [Ref - 29], including all existing domains on the list, as well as the pending list [Ref - 30] that will be included in future versions of web

[103] https://hstspreload.org/#submission-requirements

browsers. It is recommended to monitor these lists, especially the latter, once a domain has been requested to be included in the list.

In short, the activation of HSTS should be understood as a forward-looking declaration by the web server and, preferably, the domain owner that HTTPS will be supported for the lifetime of the entire domain (*.domain.com). Once the domain has been added to the preloaded list, it is not easy to remove it[104].

3.7.5 Avoiding mixed HTTP and HTTPS content

The objective of the HTTPS implementation proposed by this report is for web environments, servers or applications to make use of HTTPS for all of their content, without exception.

The objective of the HTTPS implementation proposed by this report is for web environments, servers or applications to make use of HTTPS for all of their content, without exception

Web pages with mixed content [Ref - 31] (or mixed) are those that, although initially served via HTTPS, include references to content and other web resources using HTTP (i.e. a plain text connection without encryption or authentication), such as images, other web pages, frames or iframes, scripts with Javascript code, style sheets (CSS), etc. Unfortunately, a single reference that does not make use of HTTPS could be intercepted and manipulated by a potential attacker to compromise the user's HTTPS session.

Two types of mixed content are usually defined: passive or less relevant content, since it cannot interact with the rest of the web page, such as images and multimedia content, and active or more critical content (due to its ability to interact with the rest of the web page), such as scripts, HTML content and tags (iframes), style sheets (CSS), Flash or Java objects, etc. The inclusion of active content would allow a potential attacker to take full control of the session and be able to perform any action in the web environment by impersonating the victim user. Usually, this mixed active content is blocked by web browsers[105]. However, even through passive mixed content it would be possible to manipulate the user based on the contents of the images he or she views (through social engineering), send images that include exploits, or perform content sniffing attacks where an image could be interpreted by the browser or web client as a script, along with the possibility of affecting the user's privacy and tracking his or her activities.

[104] https://hstspreload.org/#removal

[105] Depending on the type of web browser used and its version.

It is recommended to review all content and code associated with the environment, application and web pages (static and dynamic) and replace all references to "http://" with "https://" (or with relative or absolute references without specifying the protocol), except for external links to other web pages (in the "href" attribute of the "<a>" anchor tag), which, with some exceptions, are not considered mixed content (as they cause the web browser to navigate to a different web page). A distinction must therefore be made between web links (to external websites outside the control of the web environment) and the inclusion of web resources, integrated in the web environment itself (both own and external). Especially relevant are contents that are loaded as part of the web page, such as style sheets (CSS), images, scripts, HTML content (iframes), etc.[106]. The process of converting all web content to HTTPS can be complex and tedious depending on the complexity and length of the website, the current content and other possible dependencies[107].

It is recommended to review all content and code associated with the environment, application and web pages and replace all references to "http://" with "https://", except for external links to other web pages, which, with some exceptions, are not considered mixed content

In addition to own references, i.e. associated with the web environment itself, special attention should be paid to references to resources belonging to third parties (such as libraries, ad services, maps, integration with social networks, or any other functionality), as there is less control over them in order to ensure the exclusive use of HTTPS. All third-party resources should be referenced via HTTPS links and their validity and integrity (against unexpected manipulations) should be verified through a new standard called SubResource Integrity (SRI)[108].

It is very important to be aware that the inclusion of third-party web services resources and components in your own web pages creates a very close link of trust between different websites. For example, the inclusion of a third party's Javascript code in the web application means that this code must be considered as your own, as any malicious action, vulnerability or possible manipulation of the code would have complete control of the web application and a direct negative effect on it.

106 https://developers.google.com/web/fundamentals/security/prevent-mixed-content/what-is-mixed-content#mixed-content-types--security-threats-associated

107 https://developers.google.com/web/fundamentals/security/prevent-mixed-content/fixing-mixed-content

108 https://www.w3.org/TR/SRI/

Likewise, the use or loading of mixed contents (especially those considered active) may not be permitted by browsers or web clients, each imposing different criteria and restrictions in this respect, which can be evaluated through the "SSL Client Test" service [Ref - 3] ("Mixed Content Handling").

It is also possible to assess the existence of mixed content in an HTTPS web environment using tools such as the Mixed Content Scan[109].

The objective of avoiding mixed HTTP and HTTPS content can be reinforced and simplified by means of CSP (see section "3.7.6. Content Security Policy (CSP)"), especially for larger and more complex web environments, and through the use of HSTS (see section "3.7.4. Forcing strict (default) use of HTTPS: HSTS").

3.7.6 Content Security Policy (CSP)

The web standard known as Content Security Policy (CSP) [Ref - 32], apart from its multiple uses to mitigate many other types of attacks associated with web technologies (e.g. Cross-Site Scripting, XSS), can be used to improve the implementation of HTTPS.

The web standard known as Content Security Policy (CSP) [Ref - 32], apart from its multiple uses to mitigate many other types of attacks associated with web technologies, can be used to improve the implementation of HTTPS

CSP is a mechanism that allows restricting the behaviour of browsers and web clients, defining a policy that facilitates controlling how the different resources that form part of a web page are obtained (or loaded), indicating which of them are allowed and from which origins. The HTTP header "Content-Security-Policy"[110] is used for its implementation. A much broader and more rigorous use of CSP[111] is recommended for the protection of web environments than the one detailed below, which only focuses on the aspects related to the use of HTTPS. The "CSP Builder" tool[112] can be used to create the CSP policy, while "CSP Analyser"[113] can be used to analyse an existing CSP policy.

[109] https://github.com/bramus/mixed-content-scan

[110] https://scotthelme.co.uk/content-security-policy-an-introduction/

[111] https://scotthelme.co.uk/csp-cheat-sheet/

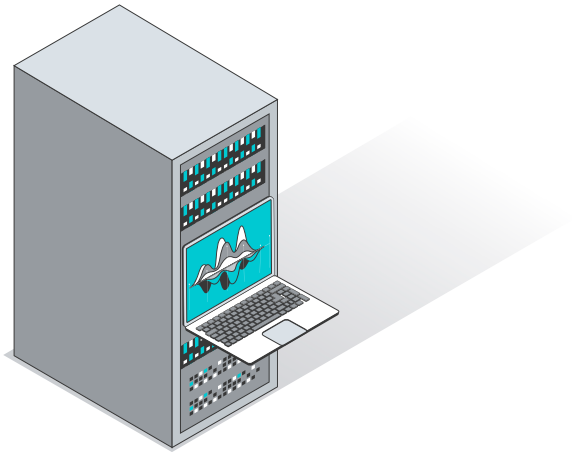
[112] https://report-uri.io/home/generate

[113] https://report-uri.io/home/analyse

CSP has evolved significantly over the last few years, initially identified by version 1.0 of the standard [Ref - 33], which is supported by most web browsers today[114] (except IE[115]). Subsequently, CSP version 2 or "Level 2" [Ref - 34], added new policies, features and protection mechanisms, some of which are associated with HTTPS and will be described later. Support for all CSP Level 2 features in modern web browsers is widespread, but more limited.114. Currently, CSP version 3 or "Level 3" [Ref - 35] is in the process of being reviewed and designed, including new policies and changes to previous versions.

In the case of HTTPS, CSP can be used to prevent the fetching of resources via HTTP links that do not use HTTPS, thus preventing the use of mixed content (see section "3.7.5. Avoiding mixed HTTP and HTTPS content").

On the one hand, CSP's "default-src" directive can be used to indicate to the web browser from where it is allowed to load the resources (including images, scripts, CSS, etc.) by default associated with the web server or application. If its value is "https:" it indicates that they can only be fetched via HTTPS, and not HTTP, regardless of the domain from which they are fetched (this simple CSP policy is not applying any restrictions in this respect). The policy can also be extended to web forms via the "form-action" directive with the same value.



Content-Security-Policy: default-src https:; form-action https:

NOTE:

In the case of CSP, as for HPKP, it is possible to strictly and effectively enforce the policy while receiving reports of policy violations (via "report-uri"). This option (described later next to "...-Report-Only") is very interesting to identify resources that should not be referenced, or attacks that have included new unauthorised web content.

[114] http://caniuse.com/#search=csp (http://caniuse.com/#feat=contentsecuritypolicy y http://caniuse.com/#feat=contentsecuritypolicy2)

[115] IE makes use of the old "X-Content-Security-Policy" HTTP header.

With the above CSP policy, the web browser will not load any resource, or submit any form, that does not make use of HTTPS. The "default-src" directive specifies the security policy to be applied by default, i.e. for all resource types that do not have directives defining a specific CSP policy for other content (fonts, images and favicons, multimedia – audio and video, child elements – such as frames and iframes, objects, scripts – Javascript, styles, connections – such as AJAX (XMLHttpRequest) and WebSockets, etc.).

With the above CSP policy, the web browser will not load any resource, or submit any form, that does not make use of HTTPS

Additionally, as with other security HTTP headers such as HPKP (see section "3.7.4. Forcing strict (default) use of HTTPS: HSTS"), and again to minimise the possible impact of incorrect configuration when starting CSP deployment, it is recommended to start by making use of CSP's reporting capabilities via the extended "...-Report-Only" header and the "report-uri" directive. It is recommended to initially test this header to identify the correct functioning of the policy. If a browser or web client receives the CSP Report-Only policy[116], and the web content references a resource that does not make use of HTTPS, it will not block HTTPS traffic (as with the standard CSP header, which does enforce the CSP policy without exception), but will generate a report directed to the URL indicated in that header by the "report-uri" directive. In this scenario, effectively, web clients are used as sensors for detecting HTTP access to resources:

```
Content-Security-Policy-Report-Only: default-src https:; form-action
https:; report-uri="https://dominioinformes.es/csp-report"
```

Unlike HPKP, in the case of CSP it is possible to use for the "report-uri" directive an HTTPS web reference associated to the same web server that has sent the CSP header (or to a different one), since in this case there will be no conflicts in the web browser to send the report (as it could happen in HPKP).

In the case of CSP it is possible to make simultaneous use of the "Content-Security-Policy" and "Content-Security-Policy-Report-Only" headers. The former would effectively enforce the current CSP policy, and the latter would allow the verification of possible violations of a new CSP policy to be introduced in the web environment.

[116] https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Security-Policy-Report-Only

It is recommended to start using the CSP header on a specific web resource (or a small set of web resources), until you verify its correct configuration, in order to avoid receiving a multitude of CSP violation reports. Unlike HSTS, CSP is a policy that is applied individually for each web page (and is not cached by the web browser), so each and every web resource to be protected must include the CSP header in its response. The impact of policy misconfiguration is minimal, as it would only affect each resource individually and only once.

The ultimate goal of migration to HTTPS is the modification of all the contents of the environment, server or web application so that all references make use of HTTPS instead of HTTP. However, this migration process can be complex, as it is necessary to update all references, both static (contained in files) and dynamic (i.e., contained in databases), to "https://". It is recommended to use relative references (such as "./directory/resource.html"), absolute references (such as "/directory/resource.html") or at least use references that do not specify the protocol (such as "//domain.com/directory/resource.html"), in order to generalise the use of HTTPS (and avoid direct references to HTTP). Using the CSP notification capabilities mentioned above can be crucial to identify resources that still use HTTP ("http://")[117].

Additionally, during the migration process (and afterwards), the "upgrade-insecure-requests" directive [Ref - 36] (associated with CSP [Ref - 33]) can be of great help. Through "upgrade-insecure-requests", the web environment will instruct web browsers that have support for this directive[118] (such as Firefox 42, Chrome 43 or Opera 30) to update any HTTP references to HTTPS, both for the web environment's own and third-party content, before sending the associated request. The only problem that may arise when using this policy is that resources that are not available via HTTPS cannot be loaded. Browsers that have support for this directive indicate this by means of the header "Upgrade-Insecure-Requests: 1" in their HTTP requests.

Content-Security-Policy: upgrade-insecure-requests; default-src https:; form-action https:; report-uri="<URL>"

It is recommended to start using the CSP header on a specific web resource (or a small set of web resources), until you verify its correct configuration, in order to avoid receiving a multitude of CSP violation reports

[117] https://scotthelme.co.uk/fixing-mixed-content-with-csp/

[118] http://caniuse.com/#feat=upgradeinsecurerequests

Another directive associated with "upgrade-insecure-requests", but more restrictive, is "block-all-mixed-content" [Ref - 31][119] (associated with CSP 3 [Ref - 35]). Using "block-all-mixed-content", the web environment will tell web browsers that support this directive not to load any content via HTTP if the web page has been fetched via HTTPS. This directive will not allow any request to be made via HTTP. In the case of using "upgrade-insecure-requests" together with "block-all-mixed-content", the latter directive will not apply, and the web browser will be more permissive in loading content via HTTP, and will at least try to obtain it via HTTPS.

Using "block-all-mixed-content", the web environment will tell web browsers that support this directive not to load any content via HTTP if the web page has been fetched via HTTPS

```
Content-Security-Policy: block-all-mixed-content;
```

NOTE:
The presence of multiple CSP headers in the same HTTP response causes the web browser to combine all received policies, restrictively applying the least common multiple of all of them[106].

In summary, and as a simple example policy focused on HTTPS, CSP can be used to restrict the use of mixed content from third parties, forcing the use of HTTPS, a scenario where HSTS cannot impose restrictions (see section "3.7.4. Forcing strict (default) use of HTTPS: HSTS"), making use of the following security policy:

```
Content-Security-Policy: upgrade-insecure-requests; default-src https: 'unsafe-inline' 'unsafe-eval'; form-action https:; connect-src https: wss:; report-uri="https://dominioinformes.es/csp-report"
```

NOTE:
The above example policy[120] specifies that any reference to AJAX (XMLHttpRequest), fetch, EventSource, and, especially, WebSockets, must also make use of HTTPS or WSS (WebSockets Secure), via "connect-src".

[119] https://w3c.github.io/webappsec-mixed-content/#block-all-mixed-content

[120] The example policy shown could be stricter, and not allow the upload and use of own content via HTTPS catalogued as "unsafe", i.e. for resources such as scripts (<script>) and styles (<style>) defined in the web page itself ("unsafe-inline"), as well as Javascript calls to "eval ()" and similar ("unsafe-eval"), removing from "default-src https:" the 'unsafe-inline' and 'unsafe-eval' directives. These directives are not directly related to HTTPS.

3.7.7 Lack of advanced functionalities in HTTP environments

Some browsers and web clients are evaluating the possibility and implementing measures to disallow access to advanced functionalities available in the browser for HTTP-based websites, and these can only be used via HTTPS (which is part of the so-called secure origins) [Ref - 37].

Among the list of advanced functionalities proposed by Chrome are those related to geolocation, orientation and movement of the user's device, access to the camera, notifications, etc.

3.7.8 Avoiding caching sensitive content

The aim of this report is that all traffic associated with web technologies should make use of HTTPS, and it is therefore necessary to differentiate between sensitive content and public content. As a general suggestion, it is recommended not to store locally on disk, or cache, on web clients any sensitive content exchanged by the web environment, server or application via HTTPS.

To do this, it is necessary to make use of different HTTP headers (with caching directives) that tell standard browsers and web clients not to store a local copy of, or cache, certain critical content[121]. The HTTP headers to be used vary depending on the browsers and web clients used to access web content (because not all of them correctly implement the standards[122]), so it is recommended to specify all of the following directives on all resources with sensitive content exchanged via HTTPS:

```
Cache-Control:  no-store, no-cache, must-revalidate, private
Pragma:         no-cache
Expires:        0
```

Directives for web content cache control are currently defined in RFC 7234 [Ref - 38].

On the other hand, public resources and content must be properly classified so that they can be cached even when using HTTPS:

```
Cache-Control:  public
```

The aim of this report is that all traffic associated with web technologies should make use of HTTPS, and it is therefore necessary to differentiate between sensitive content and public content

[121] https://securityevaluators.com/knowledge/case_studies/caching/

[122] http://stackoverflow.com/questions/49547/how-to-control-web-page-caching-across-all-browsers/5493543#5493543

3.7.9 Inspection of HTTPS traffic

Although it is globally recommended to use HTTPS only for the exchange of web content, which is the main focus of this report, there may be scenarios where it may be necessary to inspect or monitor web traffic with authorisation, for example, to detect the presence of malicious traffic or content, or attacks that could be concealed through encrypted traffic.

The inspection (or interception) of HTTPS, or TLS traffic, especially nowadays where the trend is to make widespread use of these protocols, is always controversial, with both for and against arguments. Typically, this capability is required by enterprise traffic monitoring solutions (such as intrusion detection systems, IDS) and communications networks, in particular, in organisations' internal and private networks, or in perimeter infrastructures connecting to the Internet. Its implementation can be carried out both from the network itself (using interception proxies) and/or in browsers or web clients, depending on its implementation.

Some specific scenarios where inspection of HTTPS traffic can be positive from a defensive point of view would be the identification of malware downloads, the detection of traffic directed to botnet Command & Control (C&C) servers, scenarios of data exfiltration from an organisation, etc. Without the capabilities to inspect the contents of HTTPS connections, it is only possible to obtain details of the connection metadata, i.e. the IP addresses involved, the names in the DNS and the different services used (if other than TCP/443), or to obtain certain traffic patterns. In any case, the inspection solutions should be completely transparent to the user and should not affect the other awareness and best practice behavioural recommendations that have been conveyed to users in order to identify real attacks over HTTPS[123].

Unfortunately, the widespread use of HTTPS traffic inspection solutions is widespread (between 4 and 11% of public HTTPS connections on the Internet), and in many cases their correct implementation decreases the security level of HTTPS communications (in 62% of cases), for example, by making use of weak algorithms or cryptographic suites, or even introduces additional security vulnerabilities (in 58% of cases), for example, by not correctly validating digital certificates[124].

Although it is globally recommended to use HTTPS only for the exchange of web content, which is the main focus of this report, there may be scenarios where it may be necessary to inspect or monitor web traffic with authorisation, for example, to detect the presence of malicious traffic or content, or attacks that could be concealed through encrypted traffic

[123] https://www.us-cert.gov/ncas/alerts/TA17-075A

[124] https://jhalderm.com/pub/papers/interception-ndss17.pdf

If the implementation of such a solution is being evaluated, it is recommended to carefully assess the advantages and disadvantages of such a solution. If any HTTPS traffic inspection solution is available, it is recommended to at least partially evaluate the behaviour of such a solution using the "BadSSL" service [Ref - 20].

On the other hand, inspection of HTTPS traffic is by nature contrary to the overall set of recommendations provided throughout this report, rendering the different protection mechanisms and capabilities suggested useless, and facilitating the possibility of accessing sensitive traffic that should be protected, for inspection and/or manipulation, and to impersonate the communication endpoints[125]. Traffic inspection solutions technically and effectively carry out a Man-in-the-Middle (MitM) attack (even if it is carried out with authorisation, and regardless of the traffic exchanged: personal, professional, etc.), a threat that is in fact attempted to be mitigated by the use of HTTPS.

It should be borne in mind that with the current sophistication and advances in the protection mechanisms associated with HTTPS, inspection solutions not only require modifications to the network architecture, but also to the management of HTTPS connections at both the client and server ends of the communication, otherwise activation will be difficult (e.g. if Certificate Transparency or HPKP is used, in the latter case, unless CAs are manipulated locally in browsers and web clients, both corporate and personal).

Finally, even in the definition and ratification process of the TLS 1.3 standard [Ref - 22], different groups (e.g. from the financial industry[126]) have raised concerns about the removal in TLS 1.3 of key exchange using RSA (allowing only DHE and ECDHE using forward secrecy), which eliminates the possibility of inspecting and decrypting TLS traffic in internal networks and data centres. For this reason, a new standard has even been proposed to allow HTTPS servers to be configured to use a static, rather than ephemeral, DH or ECDH key for all TLS connections, which would allow the traffic associated with these servers to be monitored[127].

If the implementation of such a solution is being evaluated, it is recommended to carefully assess the advantages and disadvantages of such a solution

[124] https://jhalderm.com/pub/papers/interception-ndss17.pdf

[125] https://blog.hboeck.de/archives/875-TLS-interception-considered-harmful-video-and-slides.html

[126] https://www.ietf.org/mail-archive/web/tls/current/msg21275.html

[127] https://tools.ietf.org/html/draft-green-tls-static-dh-in-tls13-00

3.7.10 Secure Cookies

Irrespective of whether use is made of other security mechanisms described throughout this report, such as HSTS (see section "3.7.4. Forcing strict (default) use of HTTPS: HSTS"), it is recommended to set the "Secure" directive on all cookies used by the web application so that they are only sent by web browsers and clients over an HTTPS connection, and never over HTTP.

Even when using HSTS, there are scenarios where it may be possible to obtain the value of a cookie if you do not have the "Secure" directive, for example on the first connection to the web environment, or when establishing connections to other servers in the same domain. The use of the HSTS "includeSubDomains" directive is particularly relevant when using domain cookies, whose scope and application includes the entire domain (rather than just the web application that has set the cookie value).

It is recommended to make use of the other cookie protection directives[128], even if they are not directly linked to HTTPS (e.g. HttpOnly or Same-site cookies[129]). In the case of the proposal associated with the cookie prefixes[130], both the "__Secure-" prefix and the "__Host-" prefix do require the cookie to have the "Secure" directive and to have been served via HTTPS.

Irrespective of whether use is made of other security mechanisms described throughout this report, such as HSTS, it is recommended to set the "Secure" directive on all cookies used by the web application so that they are only sent by web browsers and clients over an HTTPS connection, and never over HTTP

[128] https://www.owasp.org/index.php/Session_Management_Cheat_Sheet

[129] https://tools.ietf.org/html/draft-west-first-party-cookies-07

[130] https://tools.ietf.org/html/draft-ietf-httpbis-cookie-prefixes-00

3.7.11 Recommendations for HTTPS web content and applications

The summary of recommendations for HTTPS web content and applications includes:

- The use of HTTPS allows websites to have more relevance in the positioning (or ranking) of web search engines, such as Google.
- Search engines prioritise the indexing of web pages by default using the HTTPS version of websites, so it is recommended not to add the "noindex" directive for URLs that make use of HTTPS.
- The policy set in the "robots.txt" file must allow the indexing of the HTTPS contents of the web server by web search engines.
- It is recommended to use the "rel=canonical" tag to define that the canonical or reference resources within the web server are those that make use of HTTPS (both unique and duplicated).
- Evaluate in detail the implementation of TLS performance enhancements, such as TLS handshake (1-RTT), TLS record size and certification chain optimisations, TLS session resumption or TLS session caching (TLS session tickets), TLS False Start, TCP Fast Open, TLS early termination, OCSP Stapling, etc.
- Make use of Brotli compression mechanisms via HTTPS.
- Make use of TLS False Start together with the security mechanisms that allow it to be used with confidence (128-bit encryption, with forward secrecy and NPN/ALPN).
- Make use of TLS session resumption (or TLS session caching) by setting the maximum period for which TLS sessions will be cached, for example, to one day (24 hours). Do not share TLS session caching between different shared web hosting environments.
- Do not use TLS session tickets, and if used, use robust ticket keys, which are not frequently reused, and which are not shared between different shared web hosting environments.
- It is recommended to make use of the "preconnect" directive in the HTML <link> tag to instruct web clients to prematurely open an HTTPS connection, in order to load resources that will be referenced later.

- It is recommended to automatically redirect all HTTP traffic to HTTPS via a 301 redirect. Under no circumstances should accesses to the HTTPS version of the web environment be redirected to the HTTP version.
- Make use of HSTS (HTTP Strict Transport Security) to declare that the web server is only accessible via HTTPS.
- In the case of using HSTS, it is recommended to use a max-age value of at least 10886400 (18 weeks), and preferably 6 or 12 months ("31536000"). HSTS can be introduced by progressively increasing the max-age value.
- HSTS should be implemented for the whole domain by means of the "includeSubDomains" directive.
- The "preload" directive must be included in the HSTS configuration and the domain must be forwarded to be included in the HSTS public preload list.
- It is recommended to review all content and code associated with the environment, application and web pages (static and dynamic) and replace all references to "http://" by "https://" (or by relative or absolute references without specifying the protocol), avoiding the use of mixed content.
- Verify the validity and integrity of third party resources through SubResource Integrity (SRI).
- Make use of Content Security Policy (CSP) to mitigate the risk of mixed content usage by indicating that by default all resources should use HTTPS. Alternatively, make use of CSP's notification capabilities to identify the use of web resources that continue to use HTTP.
- It is recommended to start with a limited use of the CSP header on a specific resource, and progressively increase its use on other web resources.

- In particular, the CSP policy can be enhanced by the upgrade-insecure-requests (more permissive) and block-all-mixed-content (stricter) directives, to ensure the use of HTTPS throughout the web environment.
- It is recommended to make use of relative or absolute references to web resources, but in any case not specifying the protocol, in order to generalise the use of HTTPS (and avoid direct references to HTTP).
- It is recommended to prevent web clients from caching locally on disk any sensitive content exchanged by the web environment over HTTPS, by making use of different HTTP headers with caching directives.
- On the other hand, public resources and content must be properly classified so that they can be cached even when using HTTPS.
- In the case of making use of any HTTPS traffic inspection solution, it must be verified that it is transparent to the user, not affecting the rest of good practices, and it is necessary to carefully evaluate the advantages and disadvantages of the solution, especially verifying its correct implementation so that it does not diminish the security level of HTTPS communications or introduce additional security vulnerabilities.
- It is recommended to set the "Secure" directive on all cookies used by the web application so that cookies are only sent over an HTTPS connection.
- It is recommended to make use of the other cookie protection directives, both those that are not directly linked to HTTPS (e.g. HttpOnly or Same-site cookies) and those that are (e.g. "__Secure-" and "__Host-" prefixes)

3.8. Vulnerabilities in HTTPS

This section provides a brief list, and a very summarised description, of the most relevant vulnerabilities and attacks (term used hereafter) suffered by HTTPS-associated protocols and implementations in recent years [Ref - 10], and referenced or mentioned throughout this report.

Some of these attacks could be mitigated by simply updating the components used in the HTTPS implementation (libraries, web servers, etc.), or by modifying an insecure configuration, while others have led to drastic changes in the design of the TLS protocol, or have rendered obsolete (and permanently insecure) versions of the protocols and/or cryptographic algorithms and suites.

Therefore, as a general recommendation, always have the most up-to-date version (which at least addresses known security vulnerabilities) of the TLS libraries, web servers and web applications used in the HTTPS implementation.

A list of relevant industry vulnerabilities and events related to HTTPS, from 1994 to the present, is available in "SSL/TLS and PKI History" [Ref - 39].

The insecure renegotiation mechanisms of TLS (2009) allow MitM and Denial of Service (DoS) attacks on the web server (see section "3.4.3. Renegotiation").

This section provides a brief list, and a very summarised description, of the most relevant vulnerabilities and attacks suffered by HTTPS-associated protocols and implementations in recent years [Ref - 10], and referenced or mentioned throughout this report

BEAST (2011) is an attack against TLS 1.0 (and previous versions of SSL) and CBC-mode encryption algorithms (AES, DES, etc.), due to the use of predictable Initialization Vectors (IVs) in block ciphers, so it is recommended to use TLS 1.1, prioritising cryptographic suites based on stream (rather than block) ciphers such as RC4 (although not currently recommended due to other weaknesses discovered in 2013 and 2015, see RFC 7465[131]) and, preferably, AES-GCM with TLS 1.2[132].

CRIME (2012) is an attack against TLS-level compression mechanisms that reveals information associated with the encrypted session, so it is recommended to disable the TLS compression capabilities of the web server[133].

LUCKY 13 (2013) is again an attack against block cipher suites with CBC, based on statistical analysis and identification of small differences in the timing of cipher operations (known as padding orac*le attacks*). Again, it is recommended to avoid the use of CBC suites and instead use GCM suites with TLS 1.2.

BREACH (2013) is an attack against compression mechanisms on top of TLS (extending the previous CRIME attack), so it is recommended to disable the web server's standard HTTP web compression capabilities (widely used) and, if not possible, to apply other more elaborate mitigation techniques[134], such as HTTP Chunked Encoding. Brotli compression capabilities can also be used instead. Another alternative is to disable the compression mechanisms of the web server or application only when receiving requests from other websites, based on the Referrer header (or if the Referrer header is not available).

TIME (2013) is an attack similar to BREACH, also focused on compression mechanisms, but for which no proof of concept or practical attack tool was published.

131 https://tools.ietf.org/html/rfc7465

132 https://blog.qualys.com/ssllabs/2011/10/17/mitigating-the-beast-attack-on-tls

133 https://blog.qualys.com/ssllabs/2012/09/14/crime-information-leakage-attack-against-ssltls

134 https://blog.qualys.com/ssllabs/2013/08/07/defending-against-the-breach-attack

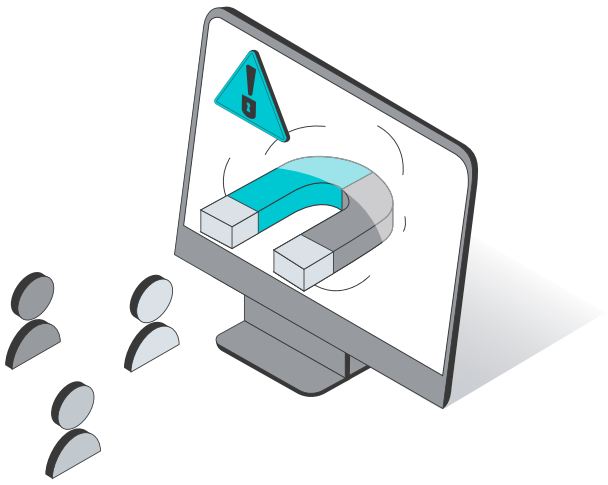
The triple **handshake** attack (2014) allows the manipulation of the TLS session when making use of client digital certificates, i.e. the (secure) renegotiation capabilities of TLS. Furthermore, this attack demonstrated how it was possible for a malicious server to select insecure DH parameters that were not even prime numbers, which would easily break the DH key exchange. Modern implementations of TLS, both client and server, have built-in defence mechanisms, so it is recommended to update all components associated with HTTPS.

Heartbleed (2014) is an attack specific to the OpenSSL library that made it possible to obtain fragments of the server's memory (in 64 KB blocks) by manipulating the TLS heartbeat functionality. Heartbleed can be mitigated by upgrading the version of OpenSSL used.

POODLE (2014) is an attack against SSL 3.0 that allows to calculate the content of an SSL encrypted connection (especially with CBC mode encryption algorithms, again, due to the difficulty of implementing them correctly), as well as to force an attack using SSL/TLS protocol version downgrade techniques, from using TLS 1.x to using SSL 3.0[135] [136].

Logjam (2015) is an attack on Diffie-Hellman (DH) parameters, and the possibility to make use of a weak key exchange via DH[137], such as 768-bit or 512-bit DH (or even, for more advanced attackers, 1024-bit DH if known DH groups are used). An attacker can force, through a MitM attack and downgrade techniques, the use of a weak cryptographic suite that allows him to decrypt the exchanged traffic. Logjam is an attack similar to FREAK, but focused on exchanging keys using DHE instead of RSA.

FREAK (2015) is an attack targeting client TLS implementations that allows intercepting HTTP communications and forcing (via downgrade techniques) weak encryption algorithms, e.g. 512-bit RSA, to be used. It affects any server that makes use of cryptographic suites allowed for export[138].



[135] https://security.googleblog.com/2014/10/this-poodle-bites-exploiting-ssl-30.html

[136] https://blog.qualys.com/ssllabs/2014/10/15/ssl-3-is-dead-killed-by-the-poodle-attack

[137] https://weakdh.org

[138] https://freakattack.com (https://censys.io/blog/freak)

DROWN (2016) is an attack against SSL 2.0 that allows decrypting HTTPS traffic by making requests to a server with SSL 2.0 using the same key[139]. The novelty of this vulnerability is that it even allows attacking servers that do not support SSL 2.0 if they reuse the same key as another server that does support this version of the SSL protocol. Despite having been published in 2016, it was unfortunately estimated that, at the time of publication, 33% of public HTTPS web servers were affected by it, due to the fact that they still provided support for SSL 2.0.

3.8.1 Recommendations of vulnerabilities in HTTPS

The summary of recommendations for vulnerabilities in HTTPS, if not already mentioned in previous sections, includes:

- Have the most up-to-date version (which at least addresses known security vulnerabilities) of the TLS libraries, web servers and web applications used in the HTTPS implementation.
- Disable the TLS compression capabilities of the web server.
- Avoid the use of block cipher suites with CBC.
- Disable the standard HTTP web compression capabilities of the web server and, if not possible, apply other more elaborate or specific mitigation techniques. Brotli compression capabilities can also be used instead.

[139] https://drownattack.com

3.9. Compatibility with web browsers and web clients

Although from a security point of view it would be desirable to reward the most modern and advanced security and protection mechanisms, a balance needs to be struck between the level of security provided by HTTPS and compatibility or interoperability with as many web clients as possible.

To this end, it is recommended, before implementing a new HTTPS-related security measure, to evaluate the existing support for it in most browsers or web clients (user agents), both traditional (e.g. Chrome, Edge, Firefox, IE, Opera, Safari, etc.) and mobile (e.g. Android web browser (<= 4.3), Chrome (Android and iOS), Safari Mobile (iOS), Firefox mobile, etc.).

Using the web service "Can I use...?" [Ref - 7] it is possible to evaluate support for a multitude of web technologies and capabilities in web browsers and web clients, such as, for example:

Different versions of the TLS protocol (1.0, 1.1., 1.2, etc.) and other TLS-specific functionalities, such as SNI (see section "3.4.2. SNI (Server Name Indication)"), directives such as "preconnect" (see section "3.7.2.3. Preconnect"), or specific cryptographic suites, such as "ChaCha20-Poly1305":	http://caniuse.com/#search=TLS http://caniuse.com/#search=preconnect http://caniuse.com/#feat=chacha20-poly1305
Support for HSTS or STS (HTTP Strict Transport Security):	http://caniuse.com/#search=hsts
Support for HPKP or PKP (HTTP Public Key Pinning):	http://caniuse.com/#search=hpkp
Support for Brotli compression:	http://caniuse.com/#search=brotli
HTTP/2 support:	http://caniuse.com/#feat=http2
Support for (different versions of) CSP (Content Security Policy) and other associated features:	http://caniuse.com/#search=csp http://caniuse.com/#feat=contentsecuritypolicy http://caniuse.com/#feat=contentsecuritypolicy2 http://caniuse.com/#feat=upgradeinsecurerequests

Additionally, through the web service "User Agent Capabilities" [Ref - 6] it is possible to identify the support of the different browsers and web clients, including also bots such as those used by Internet search engines (such as Baidu, Bing, Googlebot, Yahoo!, Yandexbot, etc.) or SSL/TLS tools and libraries (Java, OpenSSL, Tor, etc.), for the fundamental security capabilities associated with HTTPS: version 1.2 of TLS (see section "3.4.1. SSL and TLS protocol versions"), SNI (see section "3.4.2. SNI (Server Name Indication)"), forward secrecy (see section "3.5.1. Forward secrecy"), Stapling (see section "3.6.3. OCSP Stapling"),) and session tickets (see section " 3.7.2.2. TLS session resumption or caching, and TLS session tickets").

In order to assess the possible impact that a change in the HTTPS security mechanisms of the web server may have on the web clients that make use of it, it is recommended to analyse the web server logs and identify, based on the "User Agent", what percentage of web clients of a given type make use of the service, thus being able to assess and quantify the negative impact that the planned change could have.

4. Decalogue of recommendations

Security Decalogue for HTTPS Implementation

1. Configure the environment, server or web application to perform an automatic 301 redirect from HTTP to HTTPS for any web request.

   Review all content and code associated with the environment, application and web pages (static and dynamic) and replace all references to "http://" with "https://" (or with relative or absolute references without specifying the protocol), avoiding the use of mixed content.

2. Make use of HSTS (HTTP Strict Transport Security) to declare that the web server is only accessible via HTTPS.

3. Use Content Security Policy (CSP) to prevent the use of mixed content, indicating that, by default, all resources should be fetched via HTTPS. Extend the CSP policy by using "upgrade-insecure-requests" or "block-all-mixed-content" directives.

4. Use only TLS protocol, and preferably TLS versions 1.1 and 1.2 (with TLS version 1.3 coming soon).

5. Make use of 256-bit ECDSA and/or 2,048-bit RSA cryptographic keys for digital certificates (X.509).

   Carefully evaluate and select the type of digital certificate to be used, signed using SHA-256, the CA that will issue it, the renewal period, properly configure the complete certification chain, select the entity (or entities) represented by the certificate, and ensure its validity at all times.

6. Use key exchange algorithms that provide forward secrecy, preferably ECDHE (256 bits), or alternatively DHE (2,048 bits), as opposed to RSA.

   Configure the web server by prioritising the most robust cryptographic suites, listed in order of preference according to the level of security they offer and starting with "ECDHE-ECDSA-...", "ECDHE-RSA-..." or "DHE-RSA-...".

7. Make use of encryption algorithms that provide AEAD authenticated encryption, such as AES in GCM mode or ChaCha20 together with Poly1305. Symmetric encryption algorithms providing at least 128-bit security should be used. Hashing algorithms based on SHA-256 (or higher) should be used.

8. Digital certificates must include revocation mechanisms such as CRLs or OCSP or, preferably, OCSP Stapling.

   Make use of a CA that allows the OCSP Must-Staple policy to be added to digital certificates. Complementarily, make use of the notification capabilities of OCSP Expect-Staple.

9. Make use of HPKP (HTTP Public Key Pinning), or at least its notification capabilities to identify the use of illegitimate digital certificates.

   Make use of new security mechanisms for certificate issuance, such as Certificate Transparency (CT), including the Expect-CT header and using the capabilities of web servers to provide SCT responses, and of DNS CAA records.

10. Have the most up-to-date version (which at least addresses known security vulnerabilities) of the TLS libraries, web servers and web applications used in the HTTPS implementation.

ANNEX A. Requirements for the analysis of HTTPS websites

The following requirements should be taken into account by those responsible for web environments (servers, applications, etc.) that implement the HTTPS protocol, with a view to their analysis within the studies carried out by CCN-CERT to assess the implementation of HTTPS and the current state of the HTTPS ecosystem in the public sector.

The study focuses solely and exclusively on the analysis of the implementation of SSL/TLS (Secure Sockets Layer/ Transport Layer Security) on web services (HTTP and HTTPS), not extending the use of SSL/TLS to other services such as SMTPS, IMAPS, POP3S, virtual private networks (VPNs, Virtual Private Networks) etc.

Although it may seem obvious, first of all, the availability of the target web environment must be ensured at all times, verifying that it is at least available through the name resolution service or Domain Name System (DNS), and that it is at least offering HTTPS web service through the TCP/443 port (this port must always be open). Optionally, it shall be evaluated whether the target web environment is also offering HTTP web service through TCP/80 port, which can be open, filtered or closed.

The analysis will be carried out through TCP/IP connections using IP protocol version 4 (IPv4), so the target web environment must have a presence and a public IP address (IPv4) on the Internet, regardless of its presence on other networks based on IP protocol version 6 (IPv6).

The target web environment must be identified by its name or hostname (e.g. www.domain.com) and must not be identified by its IP address (e.g. 10.10.10.10) as, for example, the use of the HSTS security HTTP header is not allowed for IP addresses.

HTTPS service must not be offered over a non-standard port other than TCP/443 (e.g. TCP/444).

One of the first behaviours to be evaluated will be the existence of automatic redirections of connections established via HTTP (TCP/80) to HTTPS (TCP/443).

In addition, the validity of the certification chain or digital certificates (X.509) used by the web environment will be assessed, from the root CA certificate to the web environment's own certificate. Among other details, it will be evaluated:

- The name of the entity associated with the digital certificate and its validity with respect to the identifier of the analysed web environment (hostname).
- The validity period (or expiry period) of the digital certificate.
- The validity of the certification chain, from the root CA through the different intermediate CAs to the own digital certificate.

Additionally, multiple technical details associated with HTTPS of the target web environments will be evaluated, such as the state of implementation of standards like HTTP Strict Transport Security (HSTS), HTTP Public Key Pinning (HPKP), OSCP Stapling, the different versions of the SSL and TLS protocols supported, etc.

The study will be carried out using proprietary scanning and analysis tools, together with the evaluation carried out using the "SSL Server Test" service [Ref - 4] of Qualys SSL Labs and, in particular, using the automation APIs available in this service.

This service can also be used manually by those responsible for web environments to self-assess the status of their web environments, from the web address "https://www.ssllabs.com/ssltest/". It is always recommended to select the option "Do not show the results on the boards" before starting the analysis so that the web environment is not publicly listed through this service.

In summary, organisations interested in participating in the studies carried out by CCN-CERT to assess the implementation of HTTPS and the current state of the HTTPS ecosystem in the public sector should identify the web environments targeted by the analysis by providing the main domain (e.g. cni.es) and identifying the web servers and applications (using the following format):

- Name of the web server (e.g. www.ccn-cert.cni.es), without including the protocol (e.g. http:// or https://) or any specific resource (e.g. www.ccn-cert.cni.es/cursos/)..

In order to compile the complete list of target web environments to be analysed during the study, a document[140] is available, to be completed by the Agency's manager, providing a separate tab for each main domain to be included in the analysis, including all the web servers of that domain that will be part of the study.

Additionally, apart from the HTTPS analysis of a specific server or website, such as your Electronic Headquarters (https://sede.ministerio.es), it is recommended to extend the use of HTTPS to the entire domain, i.e. the default web server ("www"), the domain itself and all its subdomains (e.g. https://ministerio.es, https://www.ministerio.es and/or https://<subdomain>. ministerio.es).

The study of HTTPS implementation and the current state of the HTTPS ecosystem in the public sector will focus both on assessing the overall state of HTTPS implementation in the main domain (including the default web server, "www", and its subdomains), and on the specific environments and web servers provided by those responsible for them.

[140] For example, Microsoft Excel spreadsheet allowing to collect the data of the Agency, its responsible and target web servers: "CCN-CERT_TargetWebEnvironments_HTTPS_2017_v1.0.xlsx".

ANNEX B. Recommended cryptographic suites

The following list provides a set of cryptographic suites initially recommended for the configuration of the HTTPS implementation, for both RSA and ECDSA keys, in OpenSSL format, ranked in order of preference according to their level of security and interoperability [Ref - 10][141]:

ECDHE-ECDSA-CHACHA20-POLY1305
ECDHE-ECDSA-AES128-GCM-SHA256
ECDHE-ECDSA-AES256-GCM-SHA384
ECDHE-ECDSA-AES128-SHA256
ECDHE-ECDSA-AES256-SHA384
ECDHE-RSA-CHACHA20-POLY1305
ECDHE-RSA-AES128-GCM-SHA256
ECDHE-RSA-AES256-GCM-SHA384
ECDHE-RSA-AES128-SHA256
ECDHE-RSA-AES256-SHA384
DHE-RSA-AES128-GCM-SHA256
DHE-RSA-AES256-GCM-SHA384
DHE-RSA-AES128-SHA256
DHE-RSA-AES256-SHA256

NOTES:

In the above list, AES128 and CHACHA20 are conservatively prioritised over AES256 on the assumption that not all web clients have support for AES-NI[74] (although most modern clients do). Otherwise, AES256 (in italics) could be given priority over the other options.

ECDSA keys are also given priority over RSA keys in all cases, but one could also alternate the same suite for ECDSA and then for RSA (with priority over all other ECDSA-based suites).

Alternatively, the "Mozilla SSL Configuration Generator" tool[142] can be used to generate the initial HTTPS configuration of different web servers, and can specify the desired security level (or the level of compatibility required with modern or older browsers and web clients)[71] and other associated features, including the version of the web server and OpenSSL. Once the configuration is generated, it is recommended to customise and parameterise it according to the preferences and requirements of the web environment.

141 https://github.com/ssllabs/research/wiki/SSL-and-TLS-Deployment-Best-Practices

142 https://mozilla.github.io/server-side-tls/ssl-config-generator/

ANNEX C. References

[Ref – 1]	**"Best Practice Reports (BP)". CCN-CERT.** REPORTS 2017 https://www.ccn-cert.cni.es/informes/informes-ccn-cert-buenas-practicas-bp.html
[Ref – 2]	**"HTTPS: TLS, not SSL". "You've had time to secure it: I beg you, don't read it alone". Raúl Siles (DinoSec). X Jornadas STIC CCN-CERT** PRESENTATION December 2016 https://www.ccn-cert.cni.es/documentos-publicos/x-jornadas-stic-ccn-cert/1942-p2-05-https-tls-ssl-x-jornadas-stic-ccn-cert-dinosec/file.html https://www.ccn-cert.cni.es/xjornadas
[Ref – 3]	**"SSL Client Test". Qualys SSL Labs.** WEB SERVICE https://www.ssllabs.com/ssltest/viewMyClient.html
[Ref – 4]	**"SSL Server Test". Qualys SSL Labs.** WEB SERVICE https://www.ssllabs.com/ssltest/
[Ref – 5]	**"SSL Cipher Suite Details of Your Browser". DC Sec.** WEB https://cc.dcsec.uni-hannover.de
[Ref – 6]	**"User Agent Capabilities". Qualys SSL Labs.** WEB https://www.ssllabs.com/ssltest/clients.html
[Ref – 7]	**"Can I Use...?"** WEB SERVICE http://caniuse.com
[Ref – 8]	**"Transport Layer Security (TLS) Extensions: Extension Definitions" (SNI). RFC 6066. IETF.** WEB January 2011 https://tools.ietf.org/html/rfc6066
[Ref – 9]	**"TLS has exactly one performance problem: it is not used widely enough. Everything else can be optimized.". Ilya Grigorik.** WEB https://istlsfastyet.com https://hpbn.co (https://hpbn.co/transport-layer-security-tls/#optimizing-for-tls)
[Ref – 10]	**"Bulletproof SSL and TLS". Ivan Ristic. Feisty Duck.** LIBRO (BOOK) https://www.feistyduck.com/books/bulletproof-ssl-and-tls/
[Ref – 11]	**"Let's Encrypt".** WEB https://letsencrypt.org
[Ref – 12]	**"Announcing the first SHA1 collision". Google Security Blog.** BLOG POST February 2017 https://security.googleblog.com/2017/02/announcing-first-sha1-collision.html https://shattered.io
[Ref – 13]	**"Certificate Transparency (CT)". Google.** WEB http://www.certificate-transparency.org
[Ref – 14]	**"Certificate Transparency". RFC 6962. IETF.** WEB June 2013 https://tools.ietf.org/html/rfc6962

[Ref – 15]	**"Transparency in Certificates". Google".** WEB SERVICE https://www.google.com/transparencyreport/https/ct/
[Ref – 16]	**"Expect-CT Extension for HTTP". IETF.** WEB December 2016 (Draft 01: draft-stark-expect-ct-01) https://datatracker.ietf.org/doc/draft-stark-expect-ct https://tools.ietf.org/html/draft-stark-expect-ct-01
[Ref – 17]	**"DNS Certification Authority Authorization (CAA) Resource Record". RFC 6844. IETF.** WEB January 2013 https://tools.ietf.org/html/rfc6844
[Ref – 18]	**"CAA Mandated by CA/Browser Forum". Ivan Ristic. Qualys.** BLOG POST March 2017 https://blog.qualys.com/ssllabs/2017/03/13/caa-mandated-by-cabrowser-forum
[Ref – 19]	**"Public Key Pinning Extension for HTTP". RFC 7469. IETF.** WEB April 2015 https://tools.ietf.org/html/rfc7469
[Ref – 20]	**"BadSSL".** WEB SERVICE https://badssl.com https://github.com/chromium/badssl.com
[Ref – 21]	**"SSL Server Rating Guide (SSL Server Test)". Qualys SSL Labs."** WEB 19 January 2017 (version 2009n) https://github.com/ssllabs/research/wiki/SSL-Server-Rating-Guide
[Ref – 22]	**"The Transport Layer Security (TLS) Protocol Version 1.3". RFC nnnn. IETF.** WEB March 2017 (Draft 19: draft-ietf-tls-tls13-19) https://tools.ietf.org/html/draft-ietf-tls-tls13-19
[Ref – 23]	**"Transport Layer Security (TLS) Parameters" (TLS Cipher Suite Registry). IANA.** WEB https://www.iana.org/assignments/tls-parameters/tls-parameters.xhtml
[Ref – 24]	**"HTTPS as a *ranking* signal". Google.** WEB August 2014 https://webmasters.googleblog.com/2014/08/https-as-ranking-signal.html
[Ref – 25]	**"Brotli Compression". Scott Helme.** WEB September 2016 https://scotthelme.co.uk/brotli-compression/
[Ref – 26]	**"Resource Hints". W3C.** WEB March 2017 https://www.w3.org/TR/resource-hints/#preconnect
[Ref – 27]	**"HTTP Strict Transport Security (HSTS)". RFC 6797. IETF.** WEB November 2012 https://tools.ietf.org/html/rfc6797
[Ref – 28]	**"HSTS Preload List". Google.** WEB SERVICE https://hstspreload.org https://opensource.google.com/projects/hstspreload
[Ref – 29]	**"HTTP Strict Transport Security: Full List". Google.** WEB https://www.chromium.org/hsts https://cs.chromium.org/chromium/src/net/http/transport_security_state_static.json
[Ref – 30]	**"HSTS Preload Pending List". Google.** WEB https://hstspreload.org/api/v2/pending
[Ref – 31]	**"Mixed Content (Block All Mixed Content)". W3C.** WEB August 2016 https://www.w3.org/TR/mixed-content/
[Ref – 32]	**"Content Security Policy Reference".** WEB https://content-security-policy.com/
[Ref – 33]	**"Content Security Policy 1.0". W3C.** WEB February 2015 https://www.w3.org/TR/CSP1/ https://www.w3.org/TR/2012/CR-CSP-20121115/ (November 2012)

[Ref – 34]	**"Content Security Policy Level 2 (2.0)". W3C.** WEB December 2016 https://www.w3.org/TR/CSP2/
[Ref – 35]	**"Content Security Policy Level 3 (3.0)". W3C.** WEB September 2016 (Draft) https://www.w3.org/TR/CSP3/ https://www.w3.org/TR/CSP/ https://w3c.github.io/webappsec-csp/ (Work document)
[Ref – 36]	**"Upgrade Insecure Requests". W3C.** WEB October 2015 https://www.w3.org/TR/upgrade-insecure-requests/
[Ref – 37]	**"Deprecating Powerful Features on Insecure Origins". The Chromium Projects.** WEB https://www.chromium.org/Home/chromium-security/deprecating-powerful-features-on-insecure-origins https://www.chromium.org/Home/chromium-security/prefer-secure-origins-for-powerful-new-features
[Ref - 38]	**"Hypertext Transfer Protocol (HTTP/1.1): Caching". RFC 7234. IETF.** WEB June 2014 https://tools.ietf.org/html/rfc7234
[Ref – 39]	**"SSL/TLS and PKI History". Ivan Ristic. Feisty Duck.** WEB https://www.feistyduck.com/ssl-tls-and-pki-history/
[Ref – 40]	**"OpenSSL Cookbook". Ivan Ristic. Feisty Duck.** LIBRO (BOOK) https://www.feistyduck.com/books/openssl-cookbook/

centro criptológico nacional

centro criptológico nacional

www.ccn.cni.es

www.ccn-cert.cni.es

https://oc.ccn.cni.es/


ccn-cert
centro criptológico nacional
CCN
centro criptológico nacional


PUBLICACIONES
Pd
DE DEFENSA


GOBIERNO
DE ESPAÑA
MINISTERIO
DE DEFENSA
SUBSECRETARÍA DE DEFENSA
SECRETARÍA GENERAL TÉCNICA
SUBDIRECCIÓN GENERAL
DE PUBLICACIONES
Y PATRIMONIO CULTURAL