DigiCert® SSL/TLS Best Practice Workshop Student Guide

2020-03 v1

© 2020 DigiCert, Inc. All rights reserved. DigiCert is a registered trademark of DigiCert, Inc. in the USA and elsewhere. All other trademarks and registered trademarks are the property of their respective owners.

Table of Contents

Acronyms 4
Introduction 5
SSL Overview 7
SSL & TLS 7
SSL Certificates 12
Subject 13
Certificate Extensions 13
Certificate Formats 14
Certificate Signing Request (CSR) 15
SAN & Wildcard 16
Public SSL Certificates 17
DV Certificates 17
OV Certificates 18
EV Certificates 19
Domain Validation 20
Organisation Validation 22
Extended Validation 23
How SSL Works 24
SSL Handshake 28
Authority Information Access (AIA) 30
Certificate Revocation List (CRL) 33
Online Certificate Status Protocol (OCSP) 33
SSL Protocols & Algorithms 36
RSA 36
Diffie-Hellman 38
Elliptic Curve Cryptography 39
SSL Handshake Details 39
Session Resumption 43
Forward Secrecy 44
Cipher Suites 48
SSL Risks & Vulnerabilities 51
Expired/misconfigured Certificates 51
Self-signed & Vendor Certificates 54
Attacks on SSL 56

Phishing 58
Attacks on Certificate Authorities 61
Case Studies 62
Industry Trends 64
CA/Browser Forum Requirements 64
Certificate Transparency (CT) 65
Certificate Authority Authorization (CAA) 71
Examples 72
Certificate Pinning 73
Enforcing HTTPS 76
“Always-on” SSL 80
HTTP/2 81
Signed HTTP Exchanges (SXG) 83
Implementing SXG 83
Delegated Credentials 84
Automatic Certificate Management Environment (ACME) 85
SSL/TLS Best Practice 86
Security 86
Identify 87
Remediate 88
Protect 90
Monitor 91
Performance 92
Optimize cryptography 92
Use session resumption 92
Use HTTP/2 93
Use a CDN 93
Use OCSP Stapling 93
Case Study Exercise 95
About DigiBank 95
DigiBank Q&A 95
Student Exercise 97
Appendix A: SSL& TLS History 98
Appendix B: Useful Links 100
SSL Standards 100
Validation 100

Other Protocols 100
SSL Risks & Vulnerabilities 100
Industry Trends 100
Best Practice 101

Acronyms

AES	Advanced Encryption Standard
BR	Baseline Requirements
CA	Certificate Authority
CAA	Certificate Authority Authorization
CABF	CA/Browser Forum
CDN	Content Delivery Network
CRL	Certificate Revocation List
CPS	Certification Practices Statement
CT	Certificate Transparency
DES	Data Encryption Standard
DH	Diffie-Hellman
DNS	Domain Name Service
DV	Domain Validation
ECC	Elliptic Curve Cryptography
EE	End-Entity
EV	Extended Validation
FQDN	Fully-Qualified Domain Name
GDPR	General Data Protection Regulation
HPKP	HTTP Public Key Pinning
HSTS	HTTP Strict Transport Security
HTTP	HyperText Transfer Protocol
HTTPS	HTTP Secure
ICA	Intermediate CA
ICANN	Internet Corporation for Assigned Names and Numbers
OCSP	Online Certificate Status Protocol
O/S	Operating System
OV	Organization Validation
PKI	Public Key Infrastructure
RSA	Rivest Shamir Adleman
SAN	Subject Alternative Name
SHA	Secure Hash Algorithm
SNI	Server Name Indication
SSL	Secure Sockets Layer
TLS	Transport Layer Security

Introduction

Welcome to the DigiCert® SSL Best Practice Workshop!

After completing this workshop, you will be able to:

- Describe PKI/SSL risks and vulnerabilities, current SSL industry trends and SSL certificate management best practice;
- Identify risks and areas for improvement within your SSL management infrastructure.

This training is in 4 sections:

- SSL Overview
- SSL Risks
- Industry Trends
- Best Practice

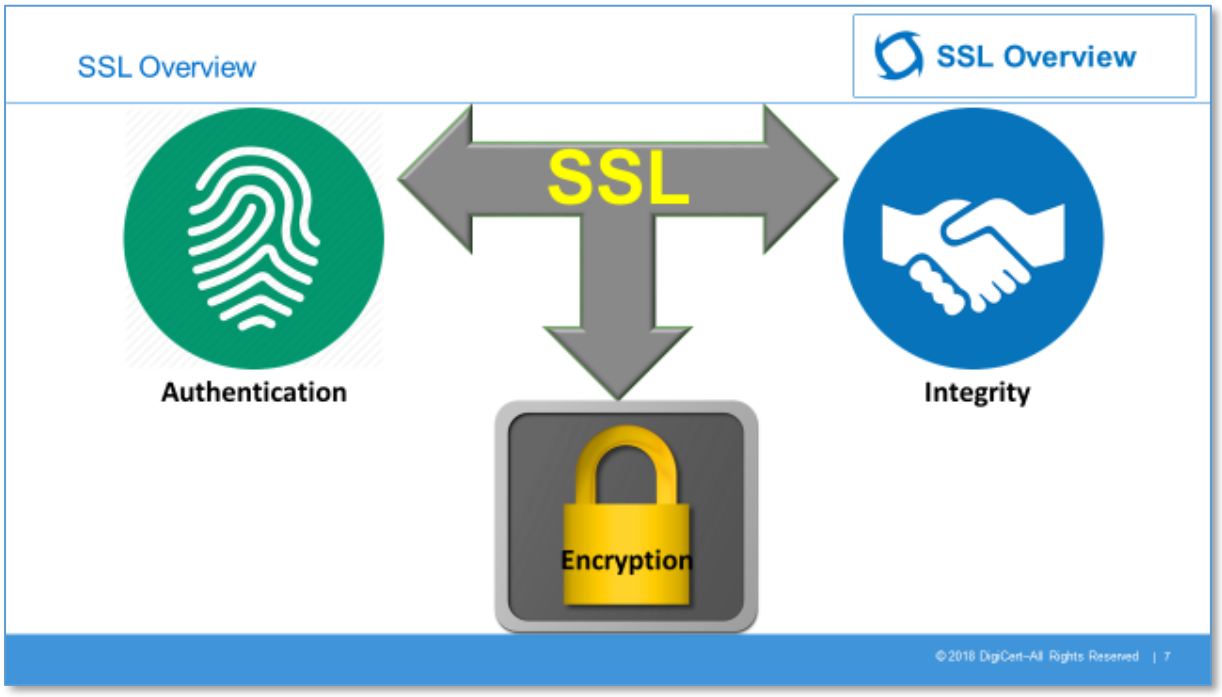
SSL Overview

SSL & TLS

The SSL (now called TLS) protocol allows client/server applications to communicate across a network in a way designed to prevent eavesdropping and tampering. TLS provides endpoint authentication and communications confidentiality over the Internet using cryptography.

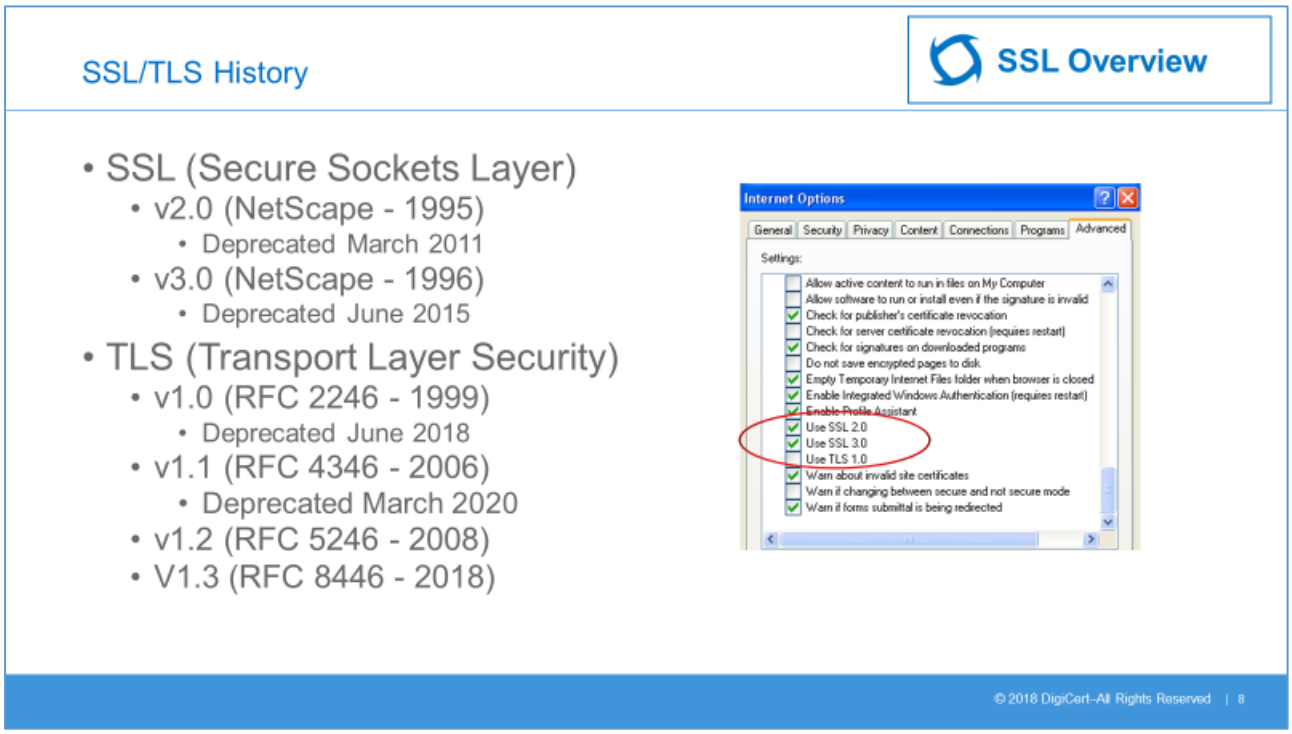
A prominent use of TLS is for securing World Wide Web traffic carried by HTTP to form HTTPS. Notable applications are electronic commerce and asset management. Increasingly, the Simple Mail

Transfer Protocol (SMTP) is also protected by TLS (RFC 3207). These applications use public key certificates to verify the identity of endpoints.



The SSL protocol is responsible for the following tasks:

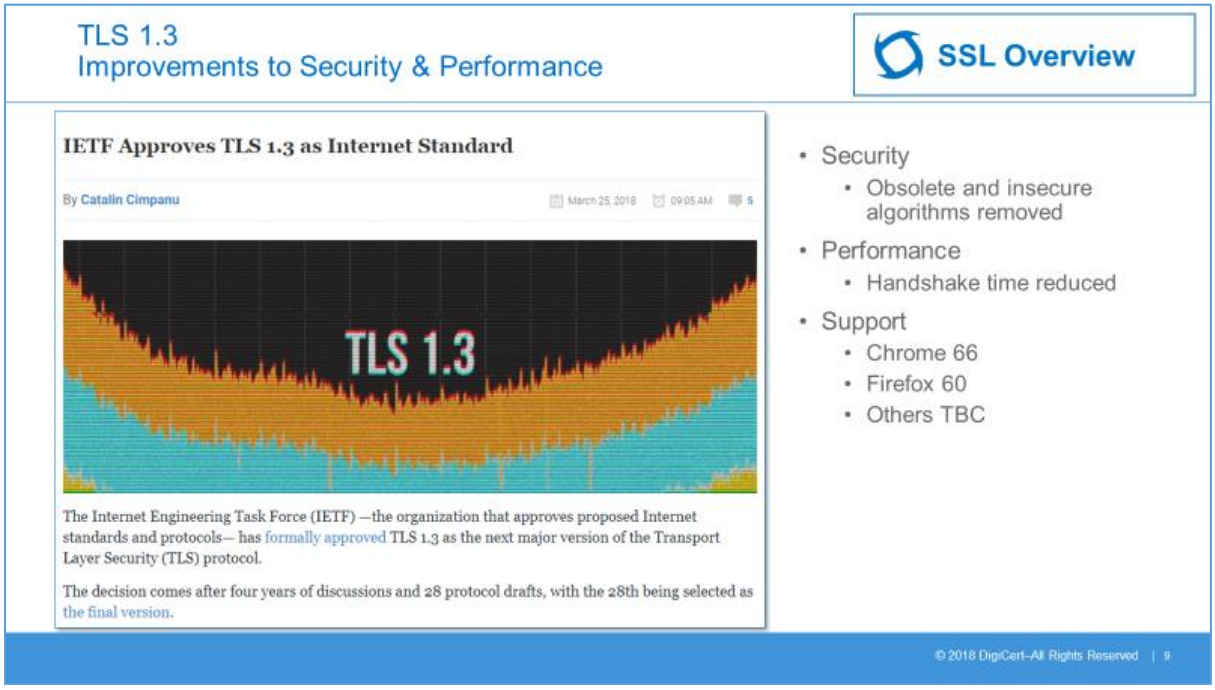
- Fragmentation of application data streams
- Compression of data before encryption
- Authentication of PDUs
- Encryption



Note: The SSL and TLS acronyms can be used interchangeably, except when referring to a specific version.

Interoperability

Since there are various versions of TLS (1.0, 1.1, 1.2, 1.3) and SSL (2.0 and 3.0), means are needed to negotiate the specific protocol version to use. The TLS protocol provides a built-in mechanism for version negotiation so as not to bother other protocol components with the complexities of version selection.



As of 26 March 2018, TLS 1.3 is an approved Internet Standard. It is based on the earlier TLS 1.2 specification.

Speed benefits of TLS 1.3

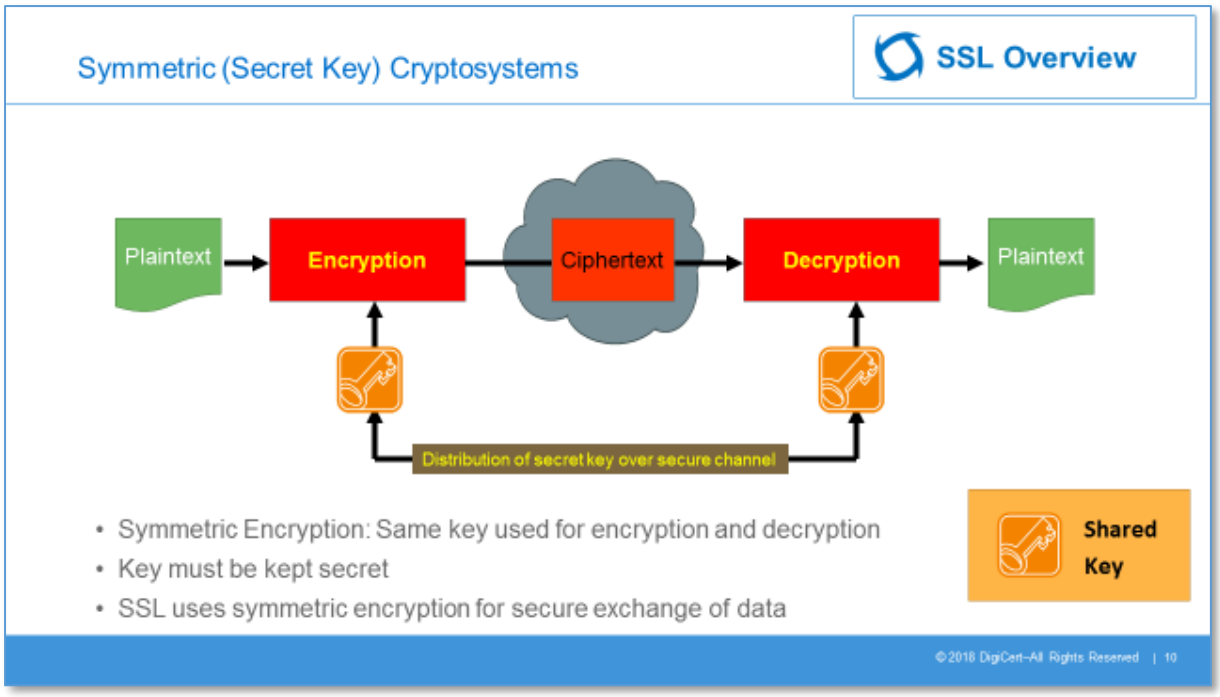
TLS and encrypted connections have always added a slight overhead when it comes to web performance. HTTP/2 helps with this problem, but TLS 1.3 helps speed up encrypted connections even more with features such as TLS false start and Zero Round Trip Time (0-RTT).

To put it simply, with TLS 1.2, two round-trips have been needed to complete the TLS handshake. With 1.3, it requires only one round-trip, which in turn cuts the encryption latency in half.

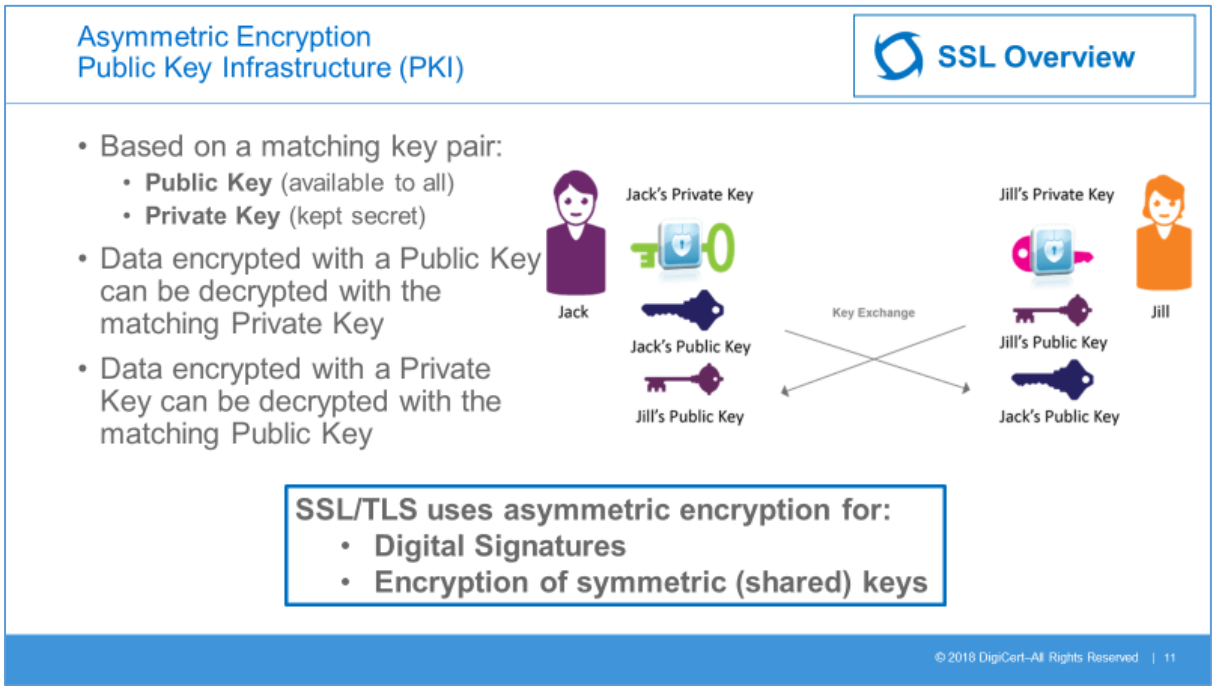
Improved security with TLS 1.3

A big problem with TLS 1.2 is that it's often not configured properly it leaves websites vulnerable to attacks. TLS 1.3 now removes obsolete and insecure features from TLS 1.2.

Because the protocol is in a sense more simplified, this makes it less likely for administrators and developers to misconfigure the protocol.



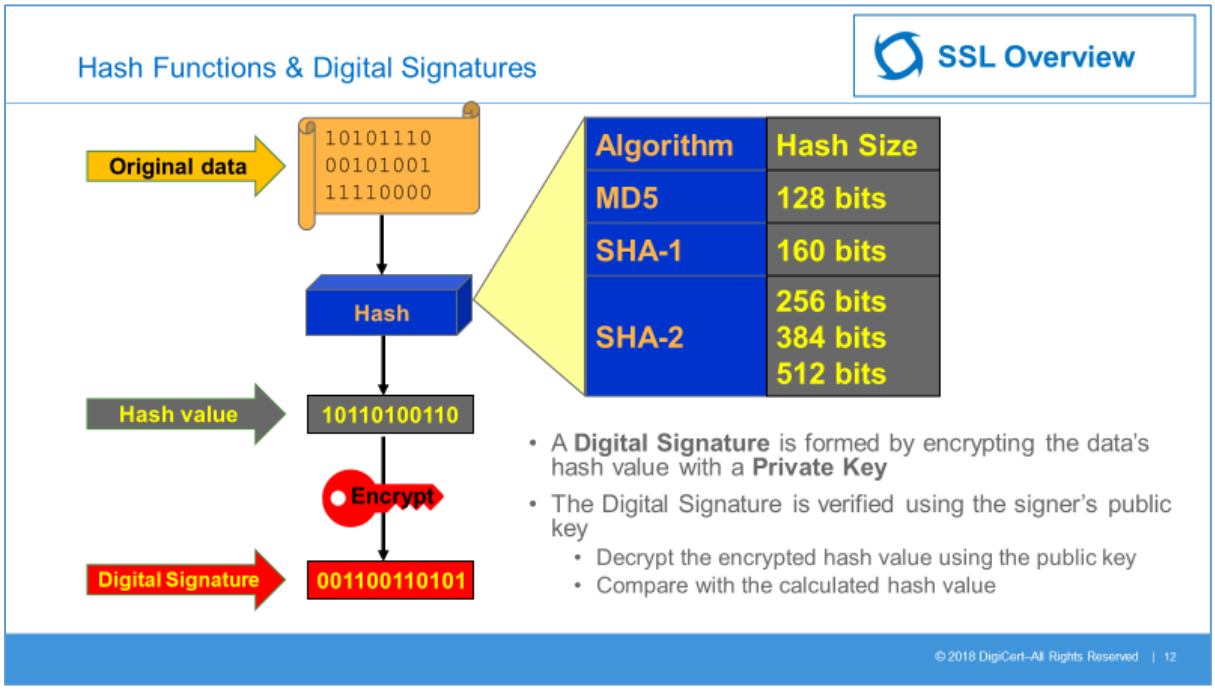
In a shared key environment e.g. for symmetric encryption, a single key is used for encryption and decryption of the data, therefore both peers need to know this shared key. The problem with this method is that there has to be a way of making sure both peers know this key without compromising the security of this key. Using the telephone or mailing the key is not really that secure, plus it is not scalable for large secure networks where there are many peer to peer sessions each with different keys. There needs to be a way of securing key exchange over an insecure path.



A digital signature is created when the sender of data creates a "hash" value for the data, encrypts the hash with his **Private Key**, and then transmits the data (plus encrypted hash). The receiver has access to the senders **Public Key** and decrypts the hash, comparing the hash value he calculates from

the data, which should come to the same value. When this happens the receiver has verified the sender's identity and also the integrity of the data sent.

SSL uses asymmetric encryption for digital signatures and (in some cases) encryption of symmetric keys.



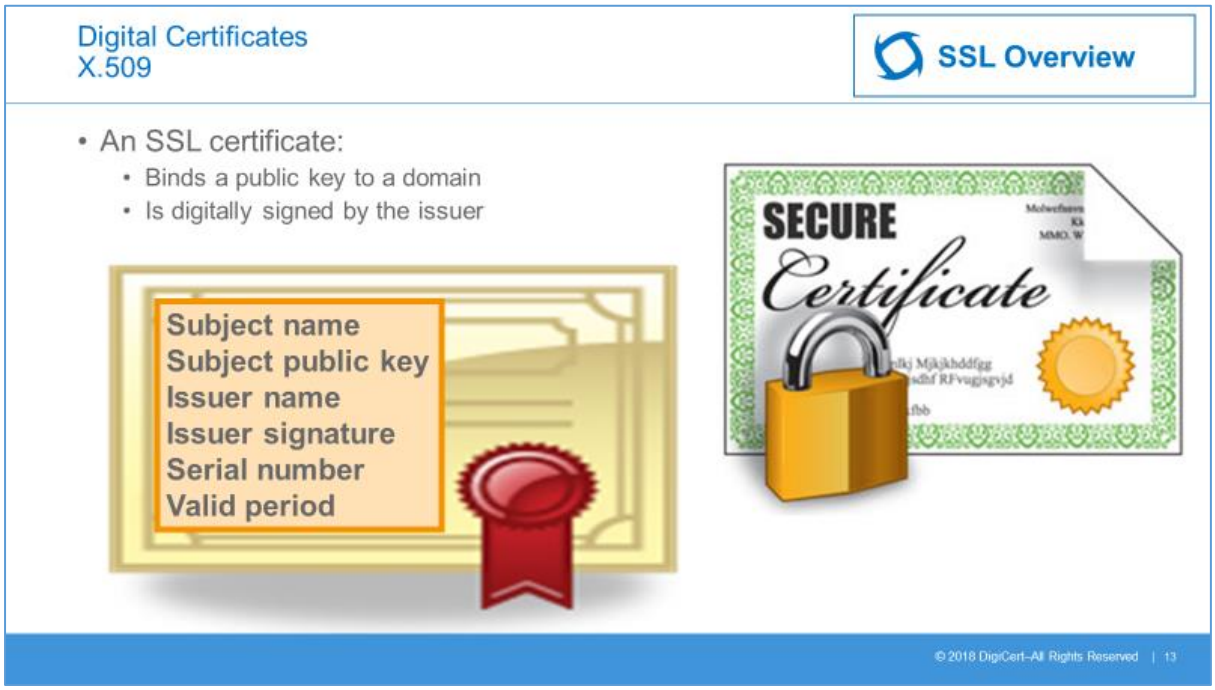
A cryptographic hash function is a mathematical algorithm that maps data of arbitrary size (often called the "message") to a bit string of a fixed size (the "hash value", "hash", or "message digest") and is a one-way function, that is, a function which is practically infeasible to invert. Well-known hash functions include MD4, MD5, SHA-1 and SHA-2

A digital signature is a mathematical scheme for verifying the authenticity of digital messages or documents.

The signature is created using the signer's private key, which is always securely kept by the signer. A hash value of the data is created and encrypted using the signer's private key. The resulting encrypted data is the digital signature.

The recipient receives a copy of the signer's public key which can be used to decrypt the signature and retrieve the hash value calculated by the sender. If this matches the recipient's calculated hash value, then they know that the data is unchanged and the identity of the sender is confirmed.

SSL Certificates



Certificates are mainly used for authentication. The subject name must match the website and the signature must be genuine.

The following fields are commonly included in an SSL certificate:

- Version
- Serial Number
- Signature Algorithm
- Issuer
- Valid From and Valid To
- Subject
- Public Key
- Subject Alternative Name (SAN)
- Basic Constraints
- Subject Key Identifier (SKI)
- Key Usage
- CRL Distribution Points
- Certificate Policies
- Extended Key Usage (EKU)
- Authority Key Identifier (AKI)
- Authority Info Access
- Logotype
- Thumbprint Algorithm
- Thumbprint

Note: with changing PKI standards, these attributes may change at any time without notice to comply with CA/B Forum requirements.

Subject

This contains the Distinguished Name (DN) information for the certificate. The fields included in a typical SSL certificate are:

- Common Name (CN)
- Organization (O)
- Organizational Unit (OU)
- Locality or City (L)
- State or Province (S)
- Country Name (C)

For Extended Validation (EV) SSL certificates, these additional fields are also included:

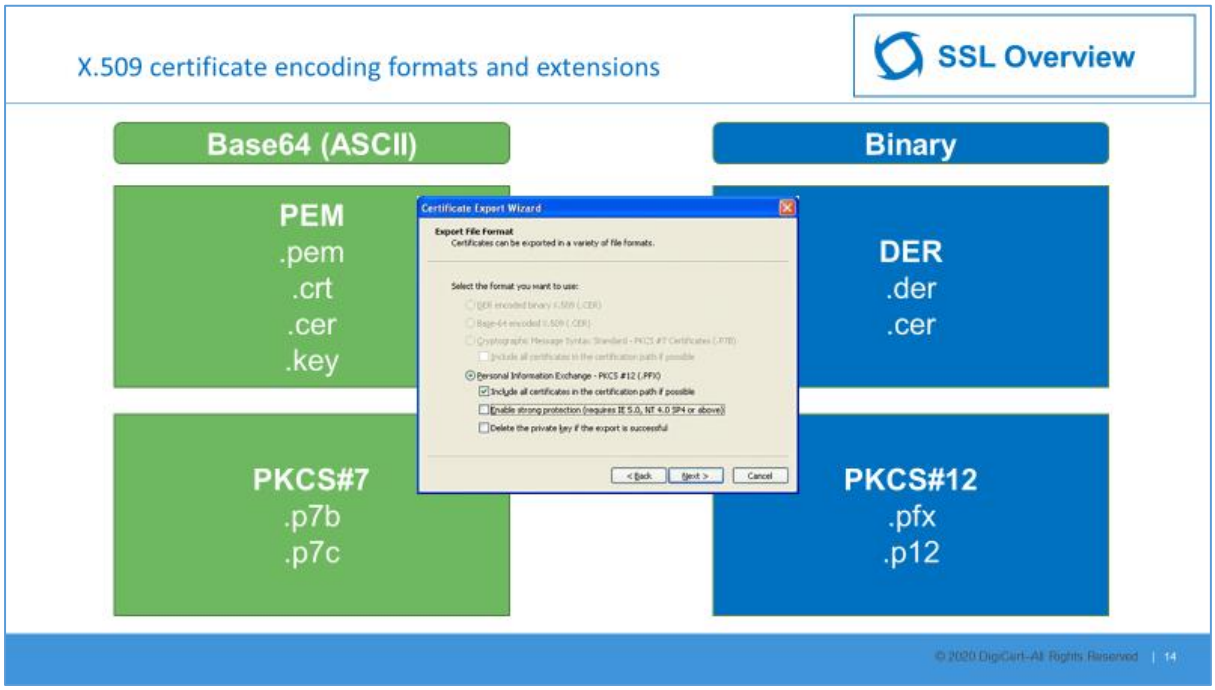
- Company Street Address
- Postal Code
- Business Category
- Serial Number (Business Registration Number)
- Jurisdiction State
- Jurisdiction Locality

Certificate Extensions

RFC 5280 (and its predecessors) defines a number of certificate extensions which indicate how the certificate should be used. Some of the most common are:

- Basic Constraints are used to indicate whether the certificate belongs to a CA.
- Key Usage provides a bitmap specifying the cryptographic operations which may be performed using the public key contained in the certificate; for example, it could indicate that the key should be used for signatures but not for encipherment.
- Extended Key Usage is used, typically on a leaf certificate, to indicate the purpose of the public key contained in the certificate. For example, it may indicate that the key may be used on the server end of a TLS or SSL connection or that the key may be used to secure email.

Certificate Formats



X.509 uses a formal language called Abstract Syntax Notation One (ASN.1) to express the certificate's data structure.

There are different formats of X.509 certificates such as PEM, DER, PKCS#7 and PKCS#12. PEM and PKCS#7 formats use Base64 ASCII encoding while DER and PKCS#12 use binary encoding. The certificate files have different extensions based on the format and encoding they use.

Most CAs (Certificate Authority) provide certificates in **PEM** format in Base64 ASCII encoded files. The certificate file types can be .pem, .crt, .cer, or .key. The .pem file can include the server certificate, the intermediate certificate and the private key in a single file. The server certificate and intermediate certificate can also be in a separate .crt or .cer file. The private key can be in a .key file.

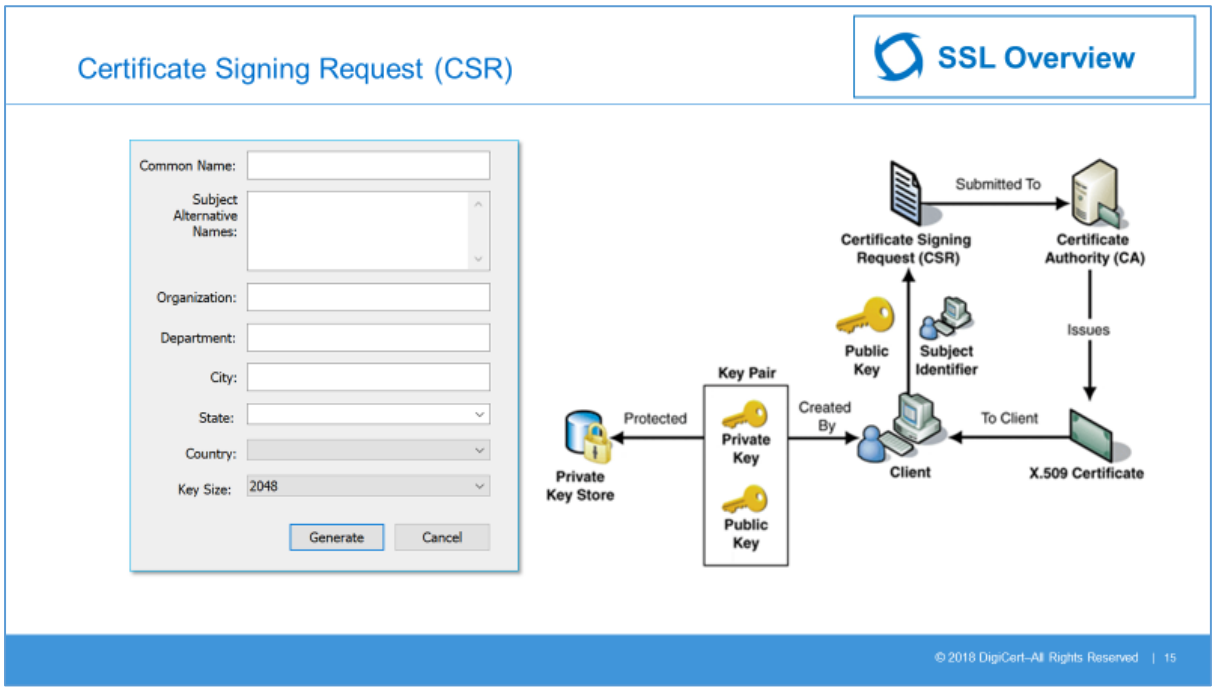
PEM files use ASCII encoding, so you can open them in any text editor such as notepad, MS word etc. Each certificate in the PEM file is contained between the ---- BEGIN CERTIFICATE---- and ----END CERTIFICATE---- statements. The private key is contained between the ---- BEGIN RSA PRIVATE KEY----- and -----END RSA PRIVATE KEY----- statements. The CSR is contained between the -----BEGIN CERTIFICATE REQUEST----- and -----END CERTIFICATE REQUEST----- statements.

The **PKCS#7** format is a Cryptographic Message Syntax Standard. The PKCS#7 certificate uses Base64 ASCII encoding with file extension .p7b or .p7c. A P7B file only contains certificates and chain certificates (Intermediate CAs), not the private key. The most common platforms that support P7B files are Microsoft Windows and Java Tomcat. The P7B certificates are contained between the "-----BEGIN PKCS7-----" and "-----END PKCS7-----" statements.

DER certificates are in binary form, contained in .der or .cer files. These certificates are mainly used in Java-based web servers. All types of certificates and private keys can be encoded in DER format.

PKCS#12 is commonly used to bundle a private key with its X.509 certificate or to bundle all the members of a chain of trust. PKCS#12 certificates are in binary form, contained in .pfx or .p12 files.

Certificate Signing Request (CSR)

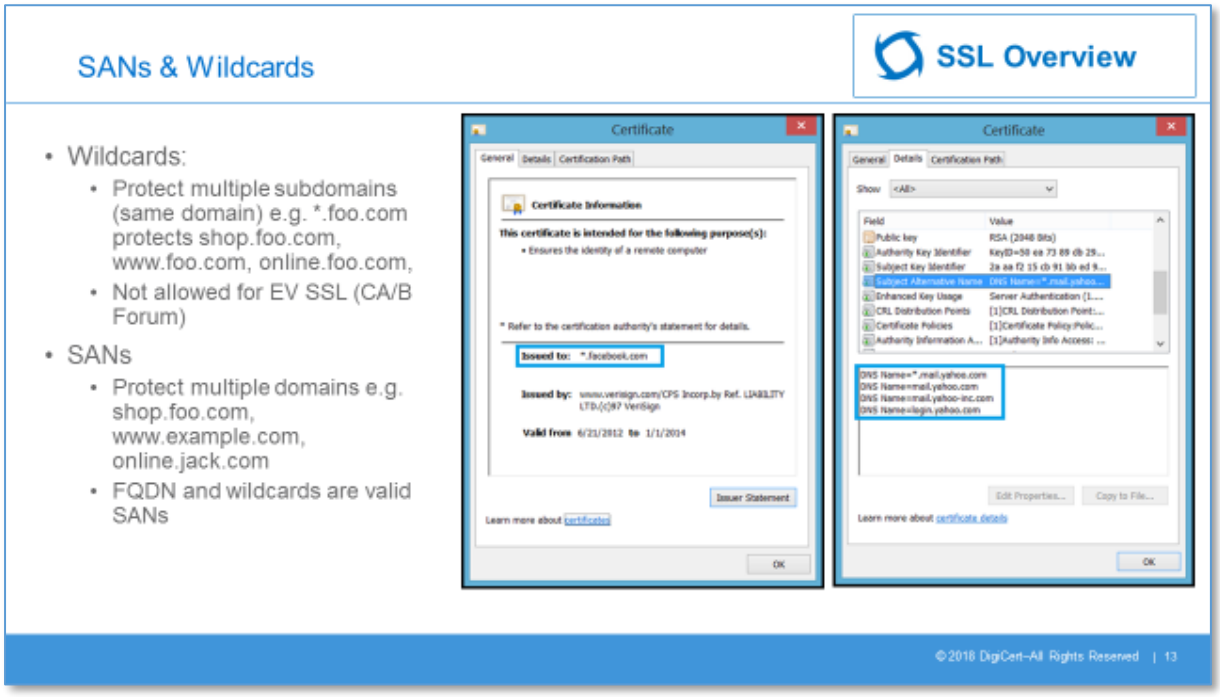


A CSR (Certificate Signing Request) is a block of encoded text that is given to a Certificate Authority when applying for an SSL Certificate. It is usually generated on the server where the certificate will be installed and contains information that will be included in the certificate such as the organization name, common name (domain name), locality, and country. It also contains the public key that will be included in the certificate. A private key is usually created at the same time that you create the CSR, making a key pair.

Most CSRs are created in the Base-64 encoded PEM format. This format includes the "-----BEGIN CERTIFICATE REQUEST-----" and "-----END CERTIFICATE REQUEST-----" lines at the beginning and end of the CSR.

A certificate authority will use a CSR to create your SSL certificate, but it does not need your private key. The certificate created with a particular CSR will only work with the private key that was generated with it. So if you lose the private key, the certificate will no longer work.

SAN & Wildcard

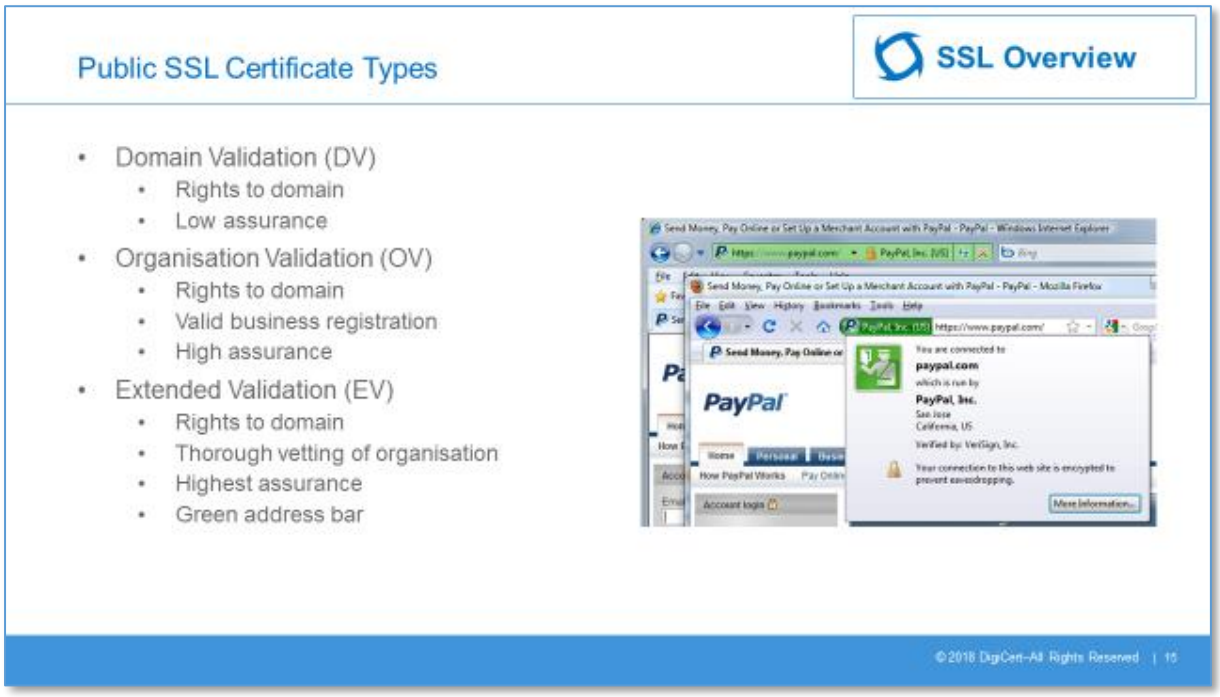


Often, SSL certificates are issued for Fully-Qualified Domains Names (FQDN) such as host.example.com. The primary hostname (domain name of the website) is listed as the Common Name in the Subject field of the certificate.

Wildcard certificates are server certificates which contain a wildcard (*) as part of the hostname. They offer a great advantage as one hostname containing a wildcard can match multiple hostnames provided they satisfy the condition.

A certificate may be valid for multiple hostnames (multiple websites). Such certificates are commonly called Subject Alternative Name (SAN) certificates or Unified Communications Certificates (UCC). SAN certificates can also contain wildcard hostnames.

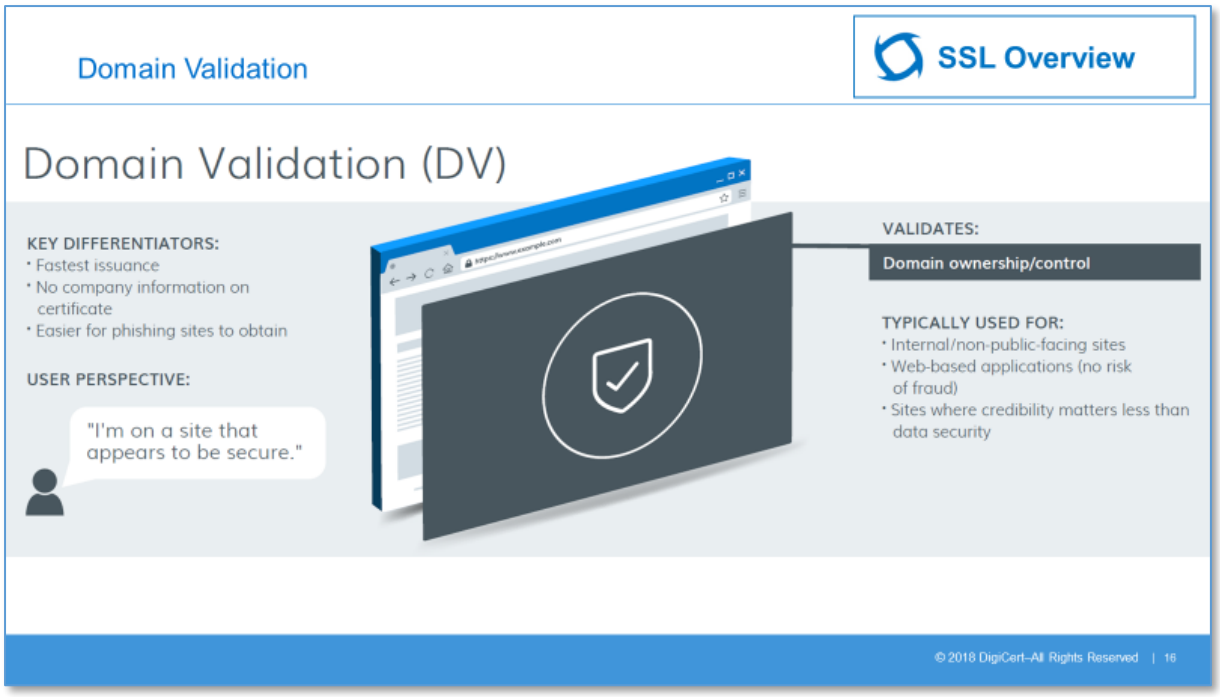
Public SSL Certificates



There are 3 categories of SSL certificates:

- Domain Validation (DV)
- Organisation Validation (OV)
- Extended Validation (EV)

DV Certificates



A low assurance or Domain-validated (DV) certificate is a certificate that only includes your domain name in the certificate (not your business or organization name). Certificate authorities usually can automatically verify that you own the domain name by checking the WHOIS record. They can be issued instantly and are cheaper but, as the name implies, they provide less assurance to your customers.

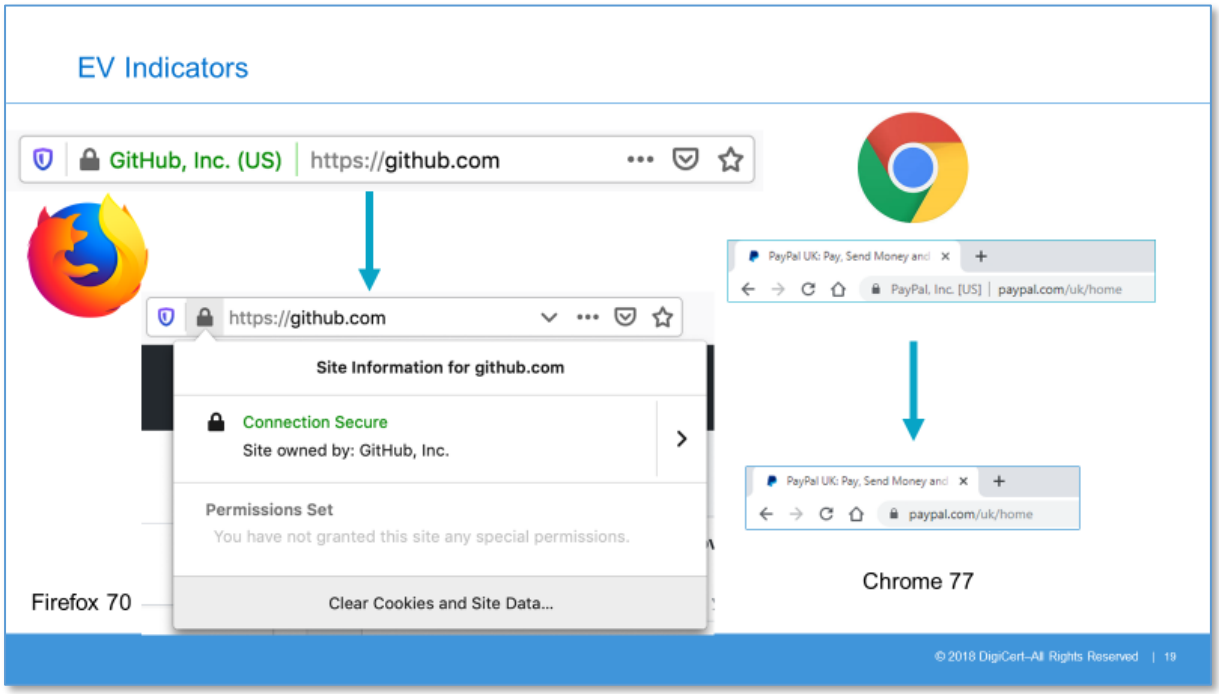
OV Certificates

A high assurance or Organization Validation (OV) certificate is the normal type of certificate that is issued. There are two things that must be verified before you can be issued a high assurance

certificate: ownership of the domain name and valid business registration. Both of these items are listed on the certificate, so visitors can be sure that you are who you say you are.

EV Certificates

An EV certificate is designed to prevent phishing attacks. It requires additional validation of your business and authorization to order the certificate. It provides even greater assurance to customers than high assurance certificates by making the address bar turn green.

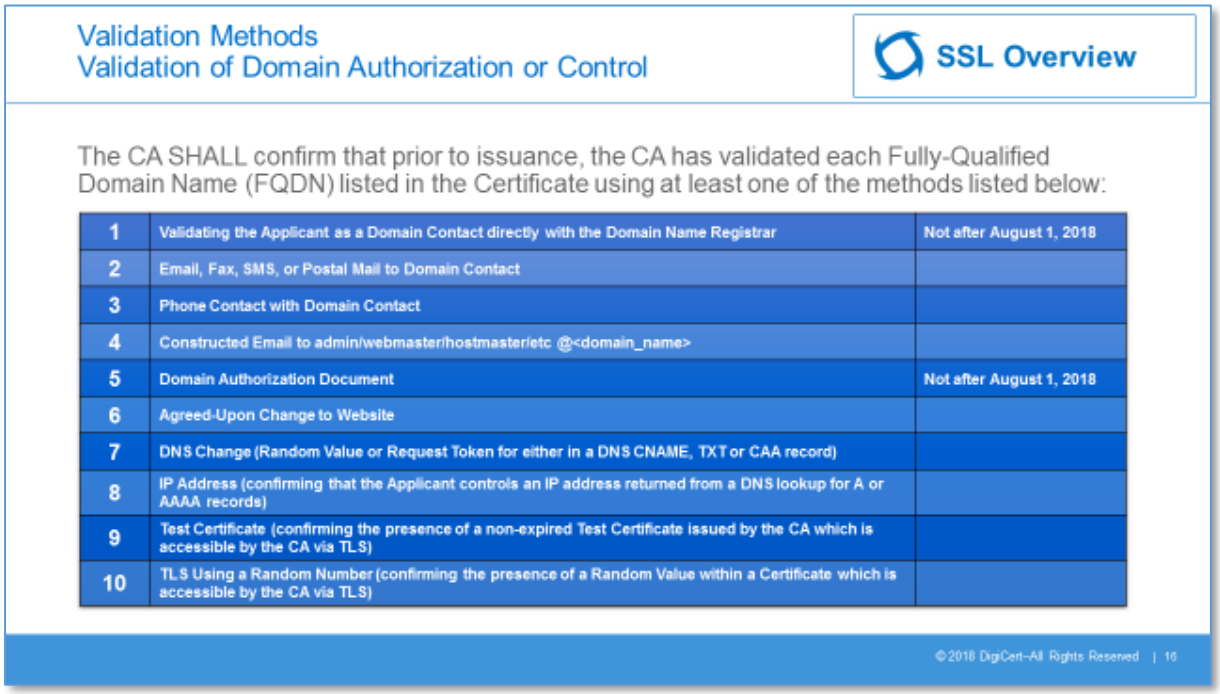


Note: Both Google and Firefox have moved the EV UI indicator out of the address bar. Both browsers will instead display the organization name in the pop-up menu that appears when you click the padlock icon (called “Page Info” in Chrome and “Identity Panel” in Firefox).

Chrome implemented this change in Chrome 77 (September 10, 2019). Firefox implemented this change in version 70 (Oct 22, 2019).

Safari and Edge browsers have already removed the organization name from the address bar. Apple have announced that the EV indicator (green bar) will be removed from Safari’s in 2020, in line with Chrome and Firefox’s removal.

Domain Validation



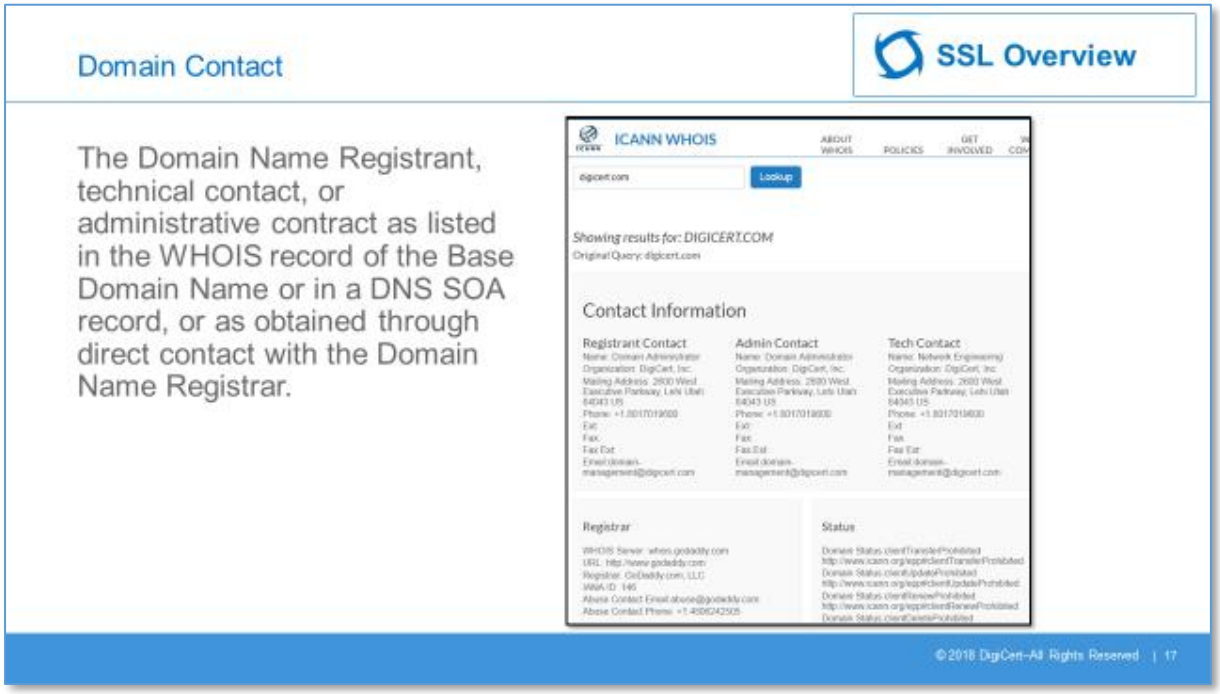
Domain verification is a critical component, regardless of certificate type, and is required for all certificates under Section 3.2.2.4 of the CA/B Forum Baseline Requirements (BRs). Domain verification confirms the certificate holder’s ownership or control over a domain name.

The CA/B Forum Baseline Requirements (BR)s permit CAs to use one of ten methods to verify domain control:

1. Validating the Applicant as a Domain Contact directly with the Domain Name Registrar
2. Email, Fax, SMS, or Postal Mail to Domain Contact
3. Phone Contact with Domain Contact
4. Constructed Email to admin/webmaster/hostmaster/etc @<domain_name>
5. Domain Authorization Document
6. Agreed-Upon Change to Website
7. DNS Change (Random Value or Request Token for either in a DNS CNAME, TXT or CAA record)
8. IP Address (confirming that the Applicant controls an IP address returned from a DNS lookup for A or AAAA records)
9. Test Certificate (confirming the presence of a non-expired Test Certificate issued by the CA which is accessible by the CA via TLS)

10. TLS Using a Random Number (confirming the presence of a Random Value within a Certificate which is accessible by the CA via TLS)

Note: CA/B Forum Ballot 218 removed methods 1 & 5 as from August 1, 2018.



The *Domain Contact* is defined as the Domain Name Registrant, technical contact, or administrative contact as listed in the WHOIS record of the Base Domain Name or in a DNS SOA record, or as obtained through direct contact with the Domain Name Registrar.

WHOIS queries are useful to identify ownership of a domain. However, spammers take advantage of this public information to call and spam you with text messages. This is why most domain registrars provide WHOIS privacy. With WHOIS privacy, the information is masked by the registrar.

With GDPR, starting on May 25th, 2018 even if you opted out of WHOIS privacy, your information is protected. This is great to avoid spam, but it can make other tasks harder since a WHOIS query can no longer identify you as the owner of a domain.

ICANN's new process allows registries and registrars to submit data to WHOIS either via a web form or an anonymized email address. For the most efficient validation process, we encourage you to let your registry and registrar know that you want them to use an anonymized email address for your domains. Doing so will ensure minimal to no impact on validation processes.

If you rely on WHOIS and still have concerns, other options are available. For example:

- Domain validation emails using any one of the following five constructed emails: admin@domain.com, administrator@domain.com, hostmaster@domain.com, postmaster@domain.com, and webmaster@domain.com.
- DNS-based validation where a token or random value is added to the TXT or CNAME record. For CNAME records, this may include a prefix to avoid changing how the domain name operates. More information is available here: https://www.digicert.com/ssl-support/validation/not-receiving-dcv-emails.htm.

- Authentication by adding a file containing a token or random value to a file at domain/.well-known/pki-validation. Confirming the random value/token is completely automated.

The CAB Forum Validation Working Group, has been exploring additional validation methods.

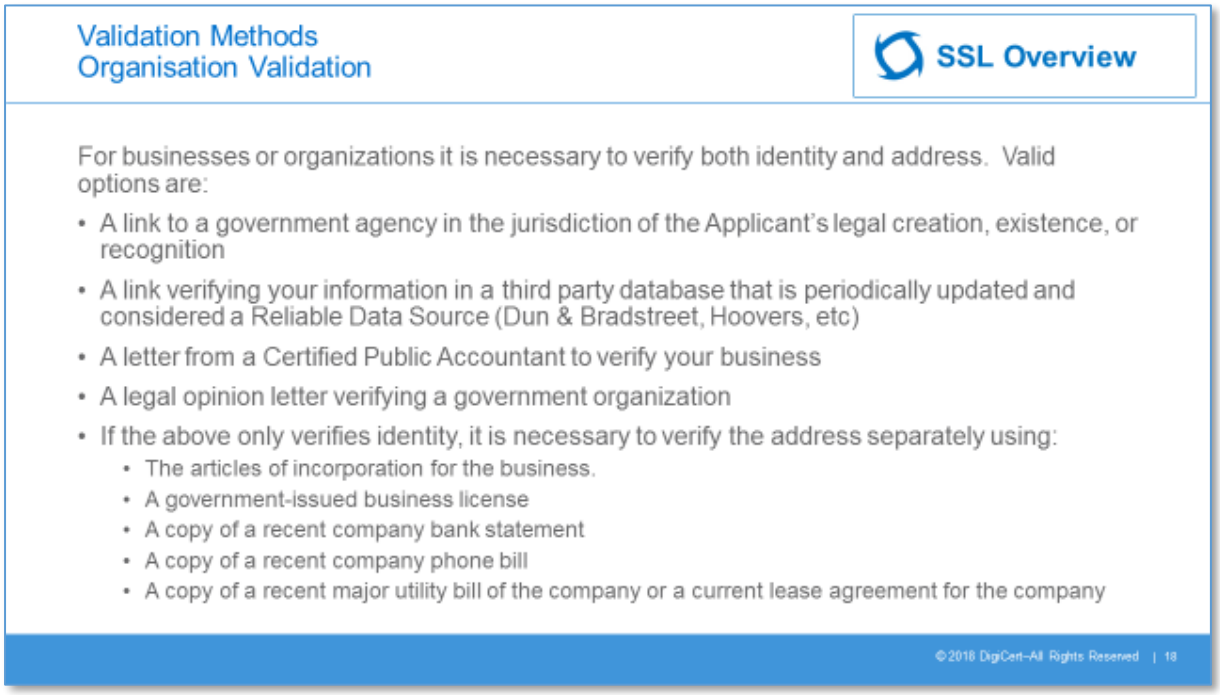
Ballot SC 13 (Dec 2018) allows customers to add public e-mail contact information in their DNS records.

https://cabforum.org/2018/12/18/ballot-sc13-caa-contact-property-and-associated-e-mail-validation-methods/

Ballot SC 14 (Feb 2019) permits domain owners to publish Domain Validation phone numbers in DNS records.

https://cabforum.org/2019/02/01/ballot-sc14-updated-phone-validation-methods/

Organisation Validation



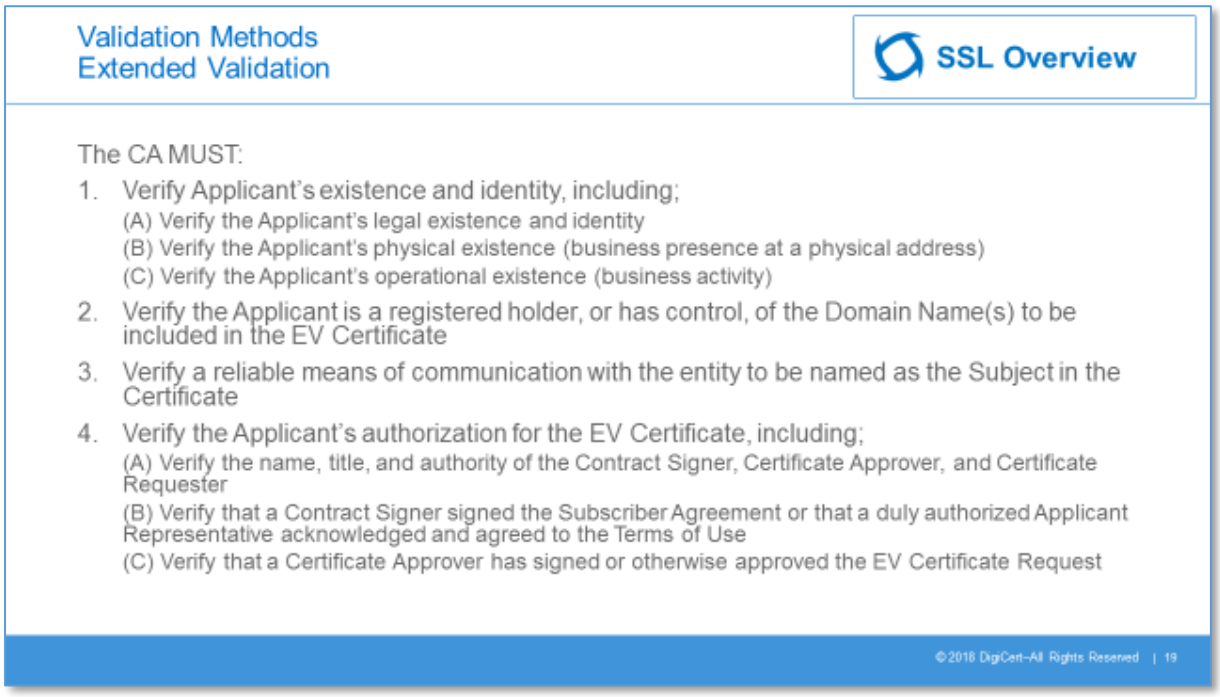
OV and EV certificates require 1) identity verification and 2) domain verification. For OV and EV, the certificate holder's identity is included in each certificate, giving transparency and clarity about the receiving party in all communications. This ensures accountability and trust.

For businesses or organizations it is necessary to verify both identity and address. Valid options are:

- A link to a government agency in the jurisdiction of the Applicant's legal creation, existence, or recognition
- A link verifying your information in a third party database that is periodically updated and considered a Reliable Data Source (Dun & Bradstreet, Hoovers, etc)
- A letter from a Certified Public Accountant to verify your business
- A legal opinion letter verifying a government organization
- If the above only verifies identity, it is necessary to verify the address separately using:
  - The articles of incorporation for the business.
  - A government-issued business license

- A copy of a recent company bank statement
- A copy of a recent company phone bill
- A copy of a recent major utility bill of the company or a current lease agreement for the company

Extended Validation



EV certificates are validated against both the CA/B Forum Baseline Requirements and the Extended Validation requirements, which place additional requirements on how authorities vet companies. These include manual checks of all the domain names requested by the applicant, checks against official government sources, checks against independent information sources, and phone calls to the company to confirm the position of the applicant. If the certificate is accepted, the government-registered serial number of the business as well as the physical address are stored in the EV certificate.

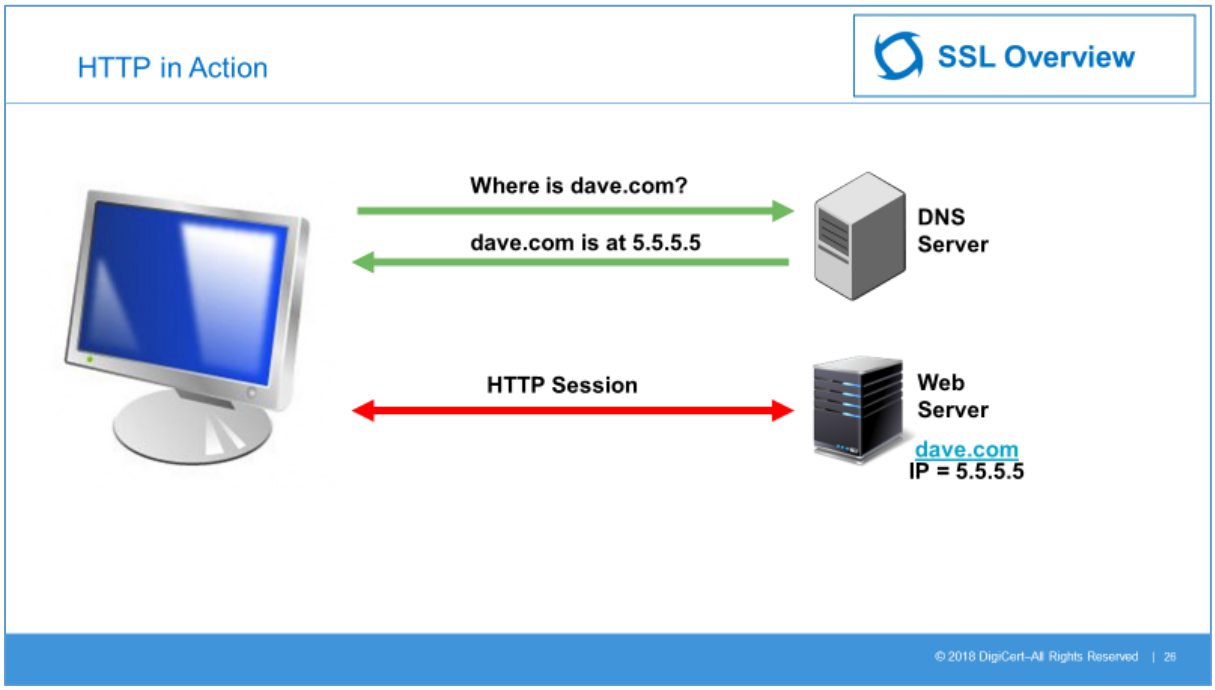
CA/B Forum Extended Validation guidelines stipulate organizations requesting Extended Validation must have their Operational Existence confirmed.

The Operational Existence requirement is satisfied if the enrolling organization has been registered and in existence for more than three years, as confirmed by the Qualified Government Independent Sources (QGIS) resource used during organization authentication.

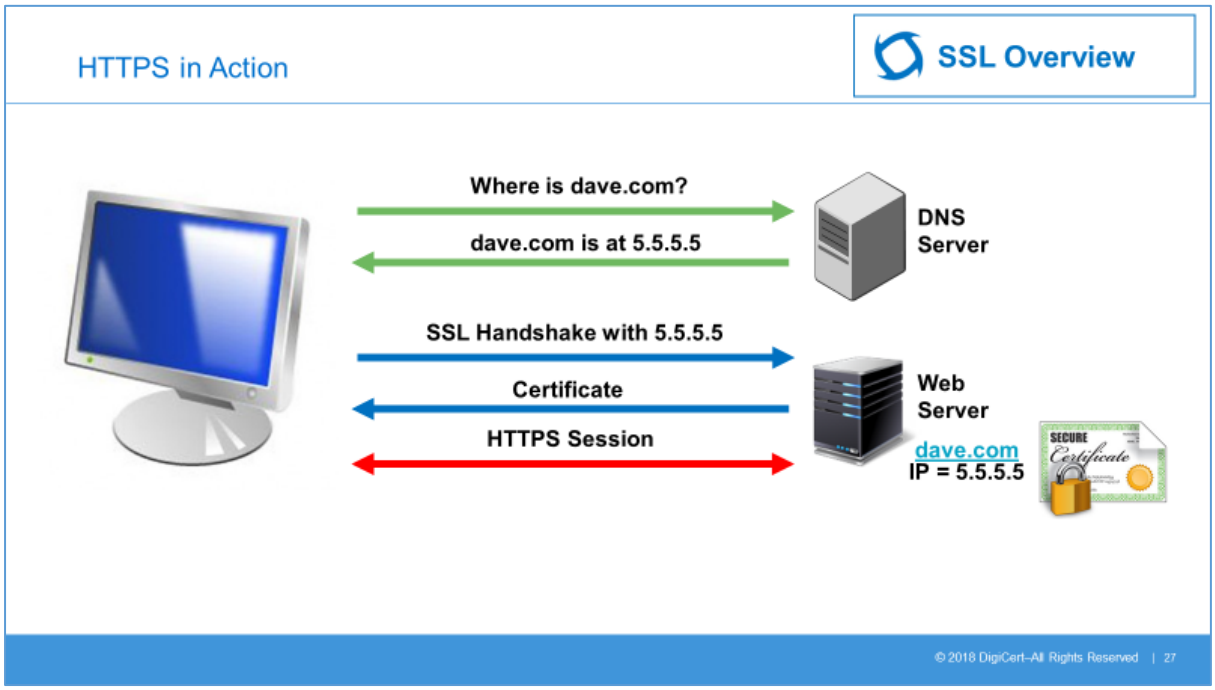
If the organization is registered for less than three years, Operational Existence can be confirmed by using other methods, for example:

- Qualified Independent Information Sources (QIIS) in examples like Dun & Bradstreet or Hoovers;
- Qualified Tax Information Sources (QTIS);
- Bank confirmation letter confirming a demand deposit account;
- Legal Opinion Letter or Attestation Letter.

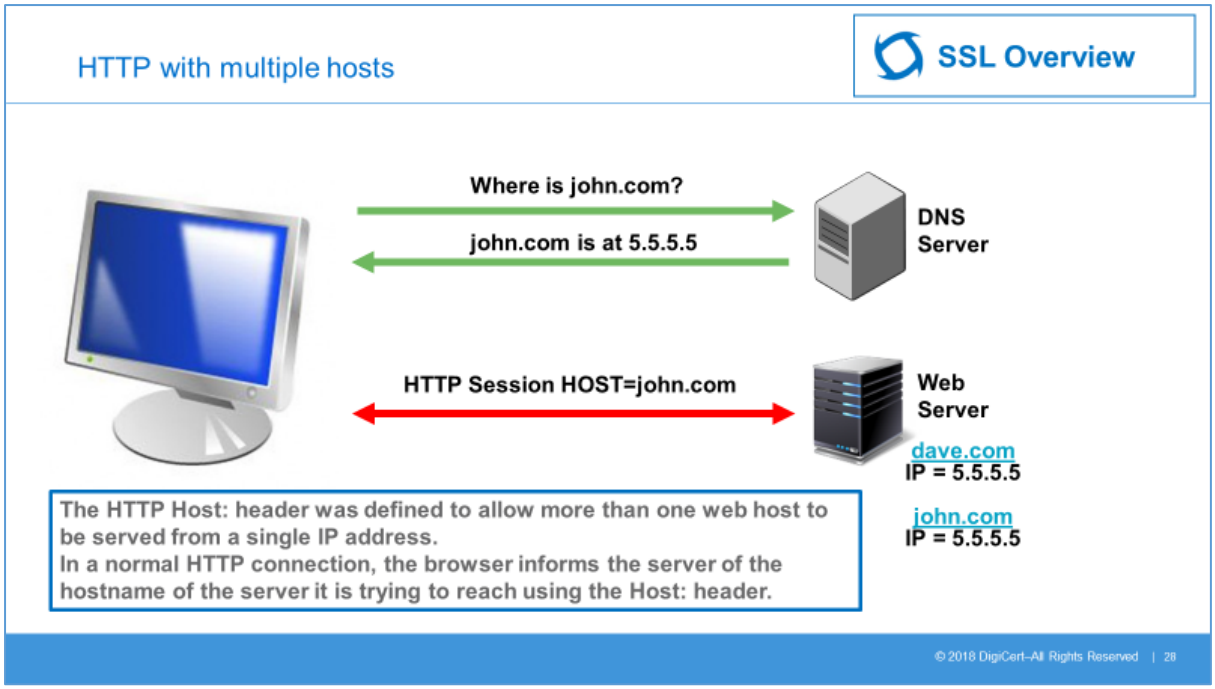
How SSL Works



The Domain Name System (DNS) is a central part of the Internet, providing a way to match names (a website you're seeking) to numbers (the address for the website). Your favourite website might have an IP address like 64.202.189.170, but this is obviously not easy to remember, so the DNS protocol is used to convert the website name to the correct IP address. Once that has been done, your browser can load the web page content using the HTTP protocol.



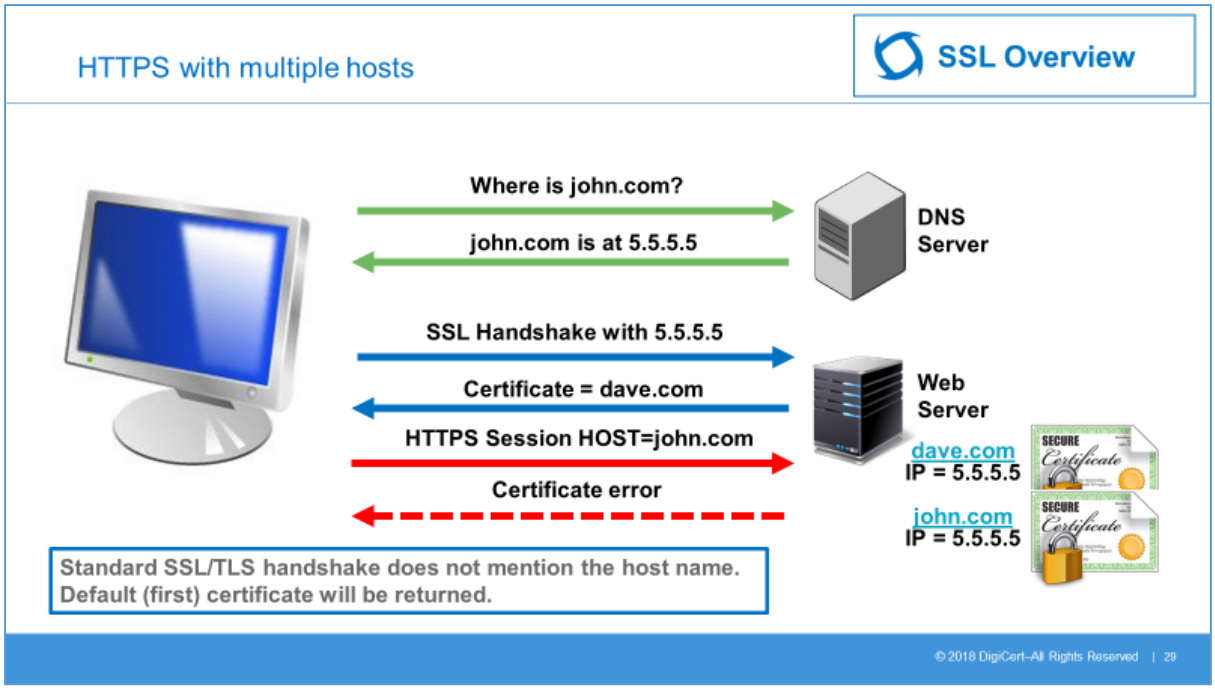
When using HTTPS, there is an intermediate step known as the SSL handshake. Once this step is completed successfully, the website has been authenticated and further communication is encrypted.



In a normal HTTP connection, the browser informs the server of the hostname of the server it is trying to reach using the Host: header. This allows for a web server on a single IP address to serve content for multiple hostnames, which is commonly known as name-based virtual hosting.

The alternative is to assign unique IP addresses for each web hostname to be served. This was commonly done in the very early days of the web, before it was widely known that IP addresses would run out and conservation measures began, and is still done this way for SSL virtual hosts (not using SNI).

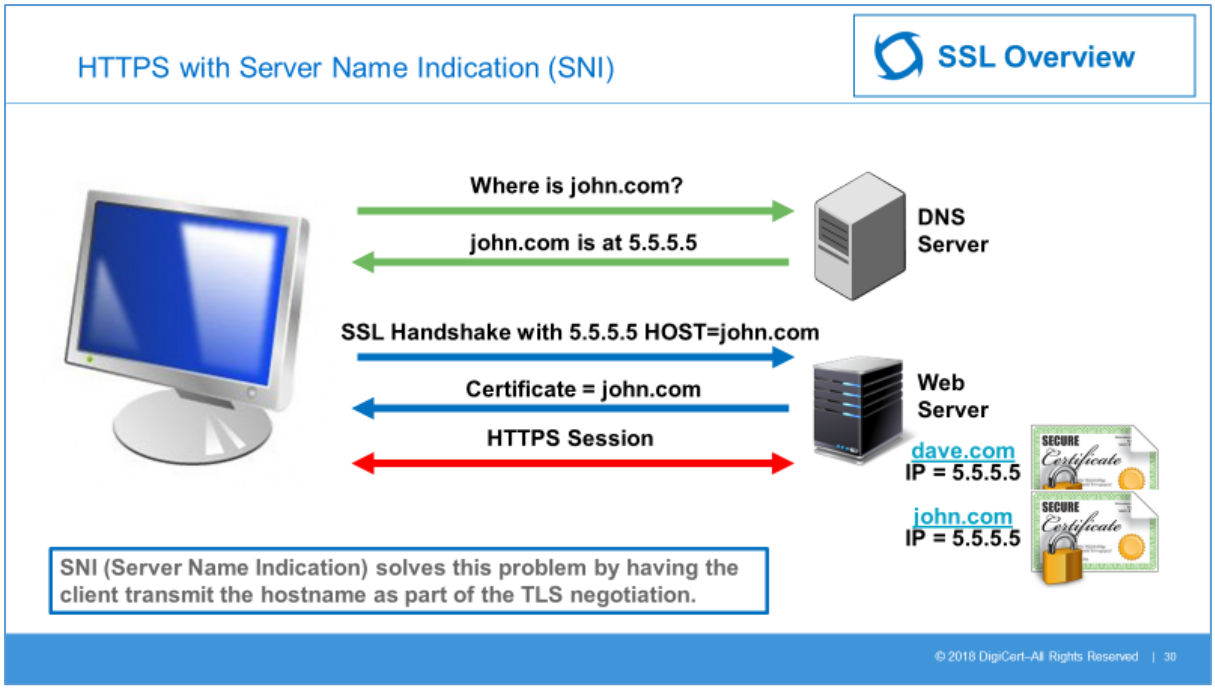
The HTTP Host: header was defined to allow more than one Web host to be served from a single IP address due to the shortage of IPv4 addresses, recognized as a problem as early as the mid-1990s. In shared web hosting environments, hundreds of unique, unrelated Web sites can be served using a single IP address this way, conserving address space.



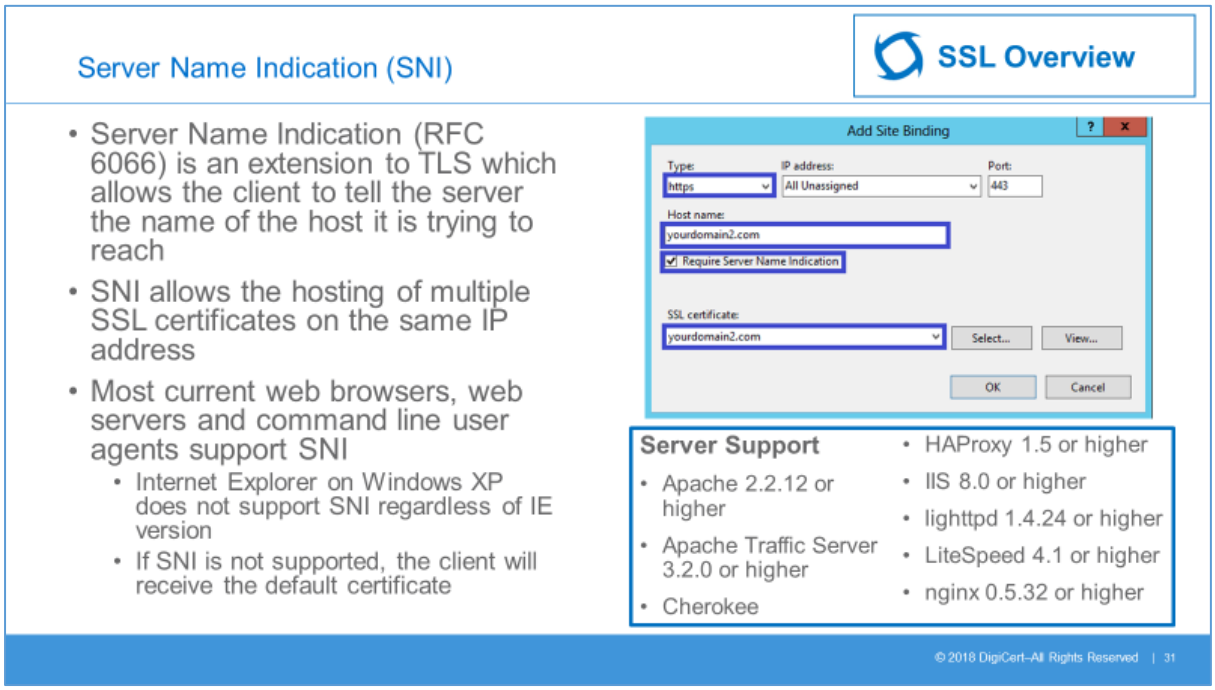
Because this method of transmitting the host name requires the connection to be already established, it does not always work with SSL/TLS connections. By the time the secure connection is set up, the web server must already know which hostname it is going to serve to the client, because the web server itself is setting up the secure connection.

It is possible for one certificate to cover multiple hostnames. The X.509 v3 specification introduced the SAN (subjectAltName) field which allows one certificate to specify more than one domain and the usage of wildcards in both the common name and SAN fields.

However it may be difficult - or even impossible, due to lack of a full list of all names in advance - to obtain a single certificate that covers all names a server will be responsible for. A server that is responsible for multiple hostnames is likely to need to present a different certificate for each name (or small group of names).



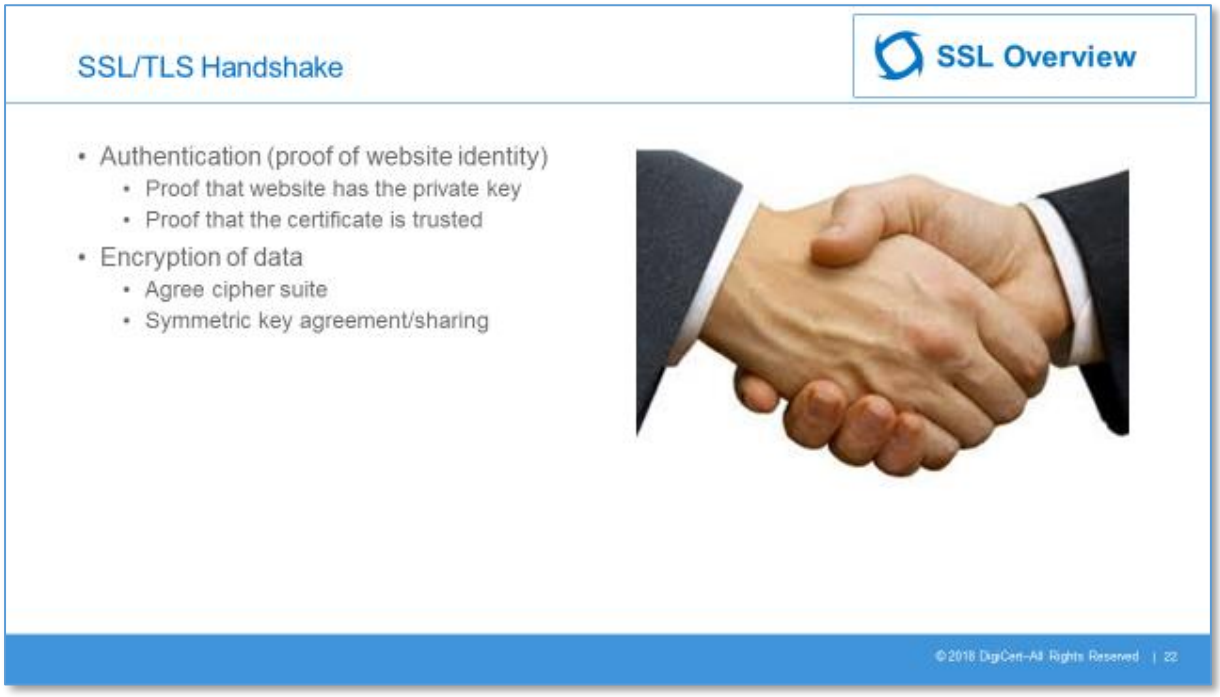
SNI solves this problem by having the client transmit the hostname as part of the TLS negotiation, so that the server is already aware of which virtual host should be used to service the connection. The server can then use the certificate and configuration for the correct virtual host.



SNI is supported by most current web browsers and web servers.

Note: Server Name Indication (SNI) exposes the hostname the client is connecting to when establishing a TLS connection. This is a potential security risk. As a consequence, **Encrypted SNI** is under investigation. Encrypted SNI keeps the hostname private when you are visiting an Encrypted SNI enabled site by concealing your browser's requested hostname from anyone listening on the Internet.

SSL Handshake



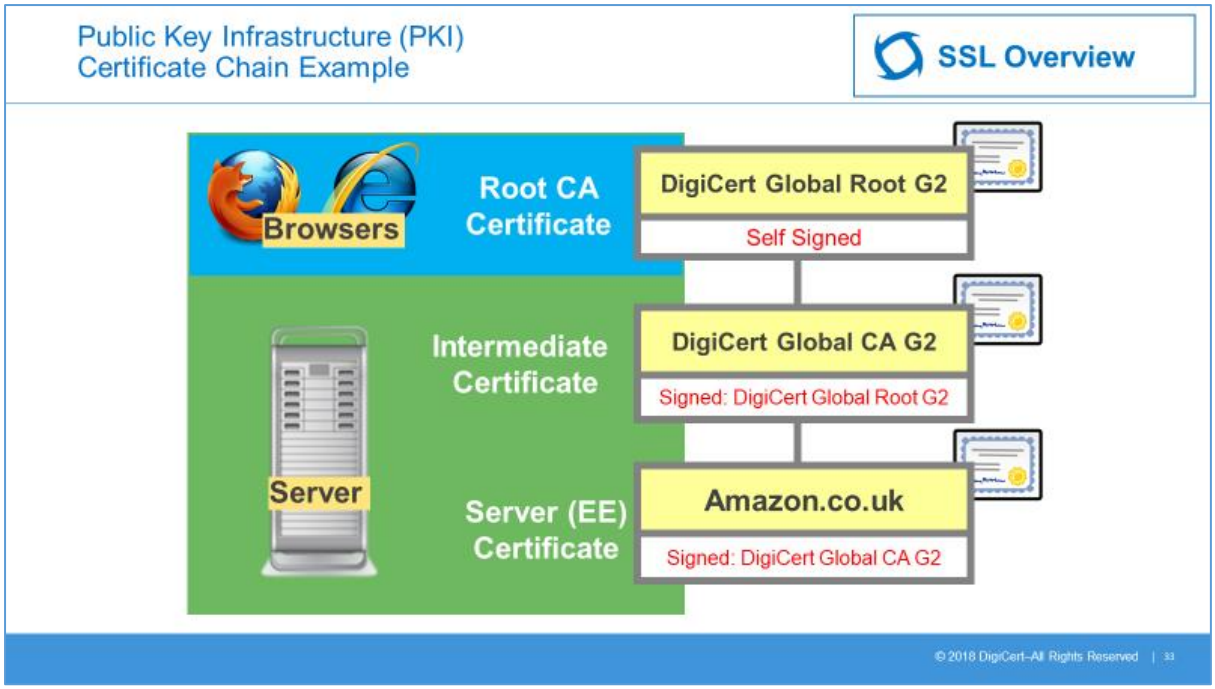
TLS has two main goals: confidentiality and authentication. Both are critically important to securely communicating on the Internet.

Communication is considered confidential when two parties are confident that nobody else can understand their conversation. Confidentiality can be achieved using symmetric encryption: use a key known only to the two parties involved to encrypt messages before sending them. In TLS, this symmetric encryption is typically done using a strong block cipher like AES. Older browsers and platforms might use a cipher like Triple DES or the stream cipher RC4, which is now considered insecure.

The other crucial goal of TLS is authentication. Authentication is a way to ensure the person on the other end is who they say they are. This is accomplished with public keys. Websites use certificates and public key cryptography to prove their identity to web browsers. And browsers need two things to trust a certificate: proof that the other party is the owner of the certificate, and proof that the certificate is trusted.

A website certificate contains a public key, and if the website can prove that it controls the associated private key, that's proof that they are the owner of the certificate. A browser considers a certificate trusted if the certificate was granted by a trusted certificate authority and contains the site's domain name.

In the context of the web, confidentiality and authentication are achieved through the process of establishing a shared key and proving ownership of a certificate. TLS does this through a series of messages called a "handshake".



Root Certification Authorities - Root CA

At the top of the hierarchical trust chain are a few Root Certification Authorities which are intrinsically trusted. Each publicly-trusted CA publishes a Certificate Practice Statement (CPS) defining on what policies user or server certificates are issued, how they are managed and how they can be revoked.

Intermediate Certification Authorities - Intermediate CA

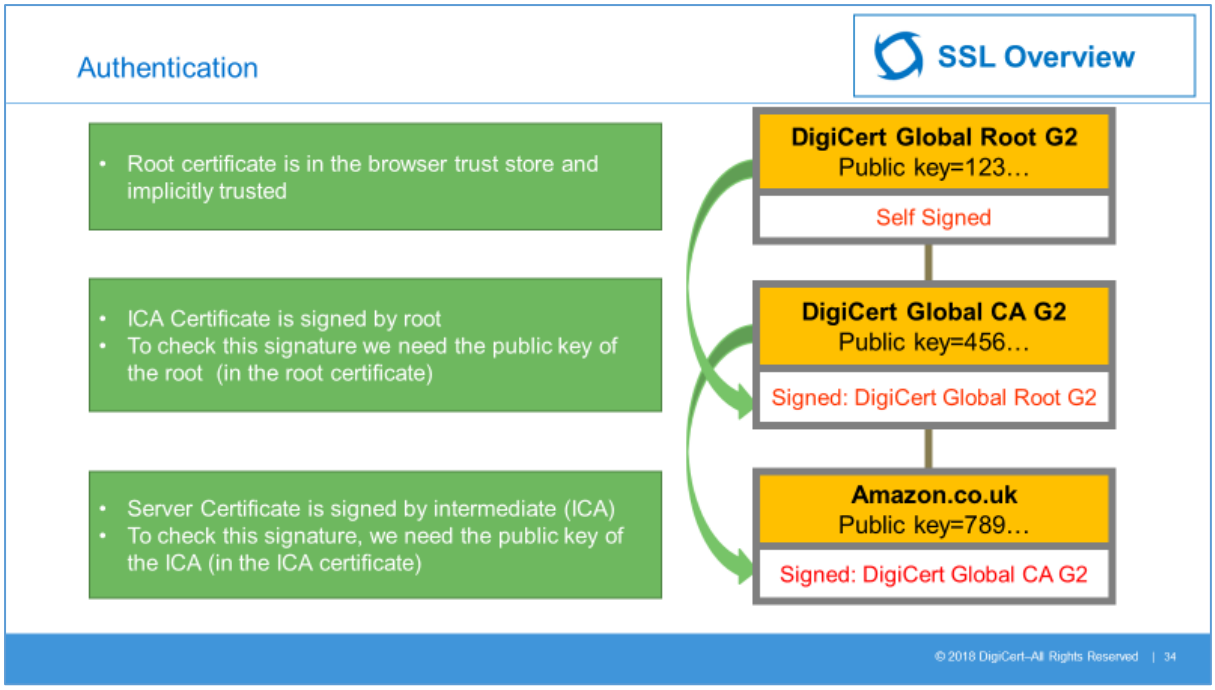
Root CAs can directly issue user certificates. This is usually done in the case of private individuals who apply directly for a certificate.

Although Root CA's can technically issue user/server certificates, this model bares significant risk because a compromised Root CA would compromise all leaf certificates issued from it.

An intermediate CA certificate is a subordinate certificate issued by the root specifically to issue end-entity server/user certificates. The result is a trust-chain that begins at the root CA, through the intermediate and finally ending with the End-Entity (server/user) certificate. Such certificates are called chained root certificates. The usage of an intermediate certificate thus provides an added level of security as the CA does not need to issue certificates directly from the CA root certificate.

The Intermediate CA (Certificate Authority) supplies the necessary chaining to a trusted root in an SSL connection and acts as a link for trust.

In principle an arbitrary number of hierarchy levels could be implemented, but usually there are not more than two or three hops from a user certificate to root CA certificate at the top of the trust chain.

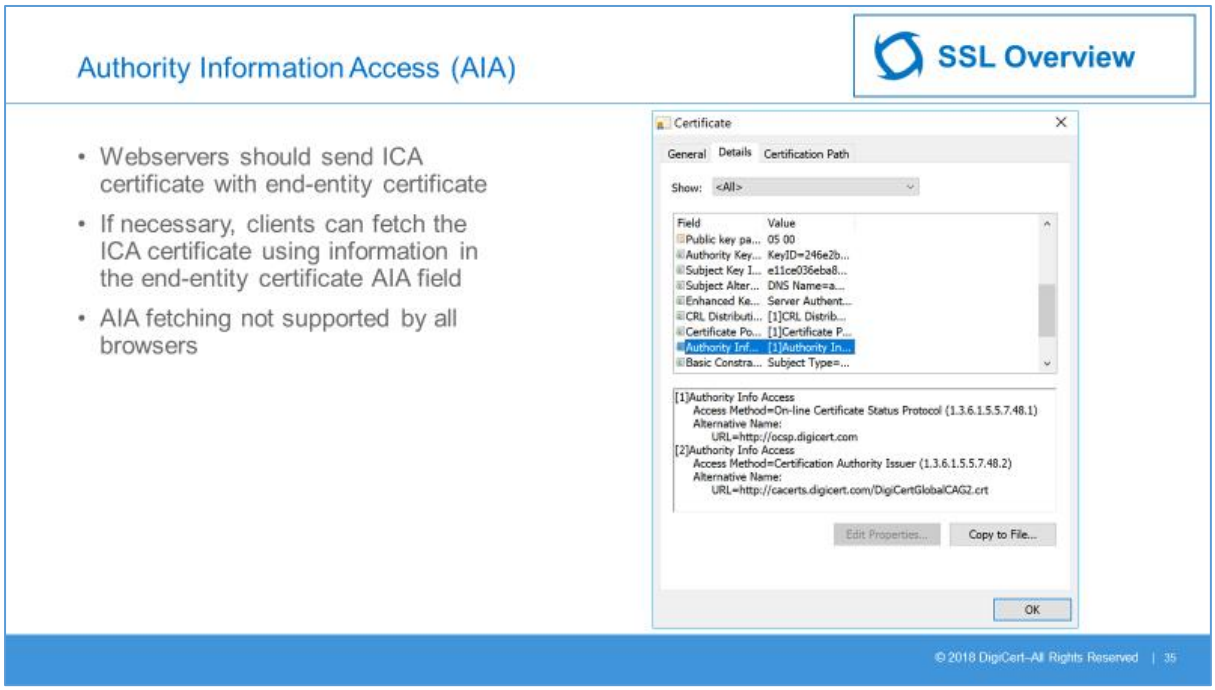


The end-entity (server) certificate contain the public key associated with the subject (domain). This certificate is signed using the private key of the Intermediate (ICA). To verify this signature, we need to use the public key of the ICA (contained in the ICA certificate).

The ICA certificate is signed using the private key of the Root certificate. To verify this signature, we need to use the public key of the root (contained in the root certificate).

The root certificate is implicitly trusted because it is found in the client's trust store. For this reason, root certificate signatures may use older signature algorithms (such as SHA-1 or even MD5) because there is no need to securely validate the root signature.

Authority Information Access (AIA)



Webservers are supposed to serve clients the end-entity certificate for their website, along with all the Intermediate certificates needed to connect to the root – this is known as the certificate chain. This allows the client to easily trace the signature back to the root certificate in its root store and verify the certificate.

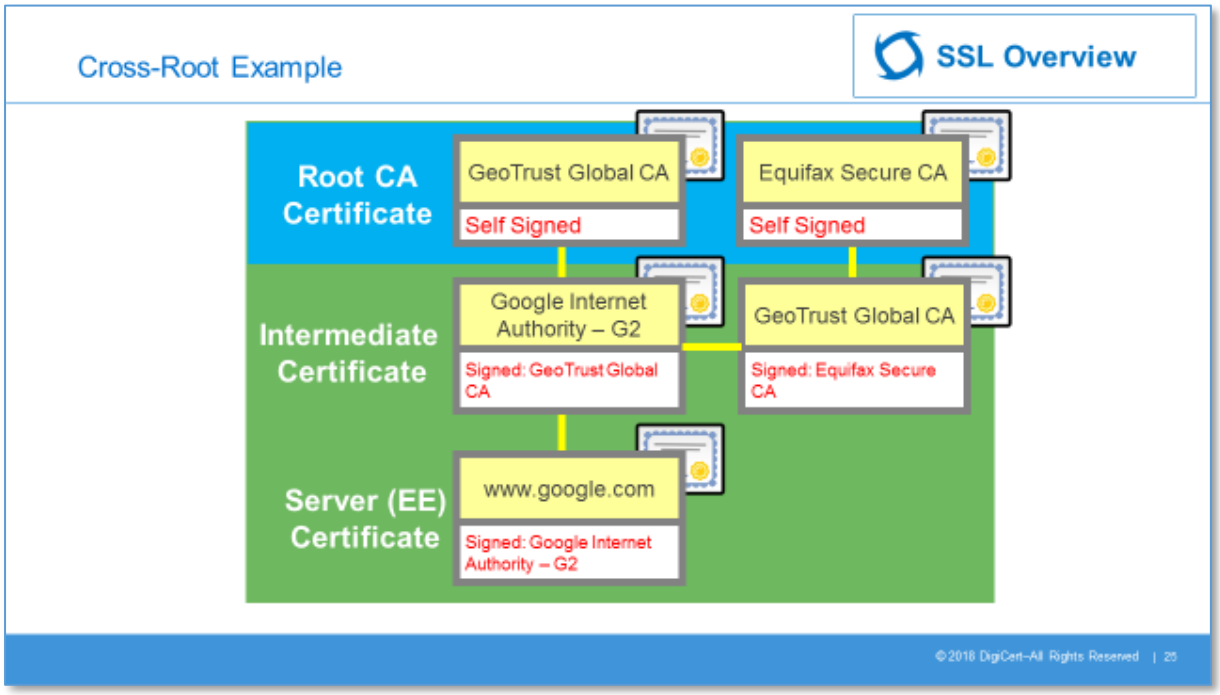
However, many webservers are not properly configured and do not provide the right certificate chain. Normally, this should be a bigger issue, given that a lot of servers have improperly configured certificate chains. But clients have a way to account for this common problem: AIA fetching.

AIA, or Authority Information Access, is an extension in SSL certificates that provides information about the issuer. One of the purposes of this extension is to provide a link to the issuing intermediate certificate. If a server does not provide the intermediate certificate, clients that perform AIA fetching will download the certificate from that URL.

Not all client software performs AIA fetching. Google Chrome on all platforms (except Android), Internet Explorer, and Safari all perform AIA fetching. However, the entire Android operating system does not do AIA fetching. Nor does Firefox.

There are other methods for solving the problem of missing intermediate certificates. Clients can cache intermediate certificates they encounter and use them for future connections. Or they can come pre-shipped with common intermediates. Some clients use a combination of these.

The underlying operating system can also handle some of these tasks. For instance, on Windows, if you visit a site with a missing intermediate in Chrome, the browser will perform AIA fetching and then Windows will cache the intermediate. If you then view the site in Firefox, it will pull the intermediate from Window's cache and everything will work. But if you had done this in reverse order, Firefox would have presented an error because it would have no way to get the intermediate.

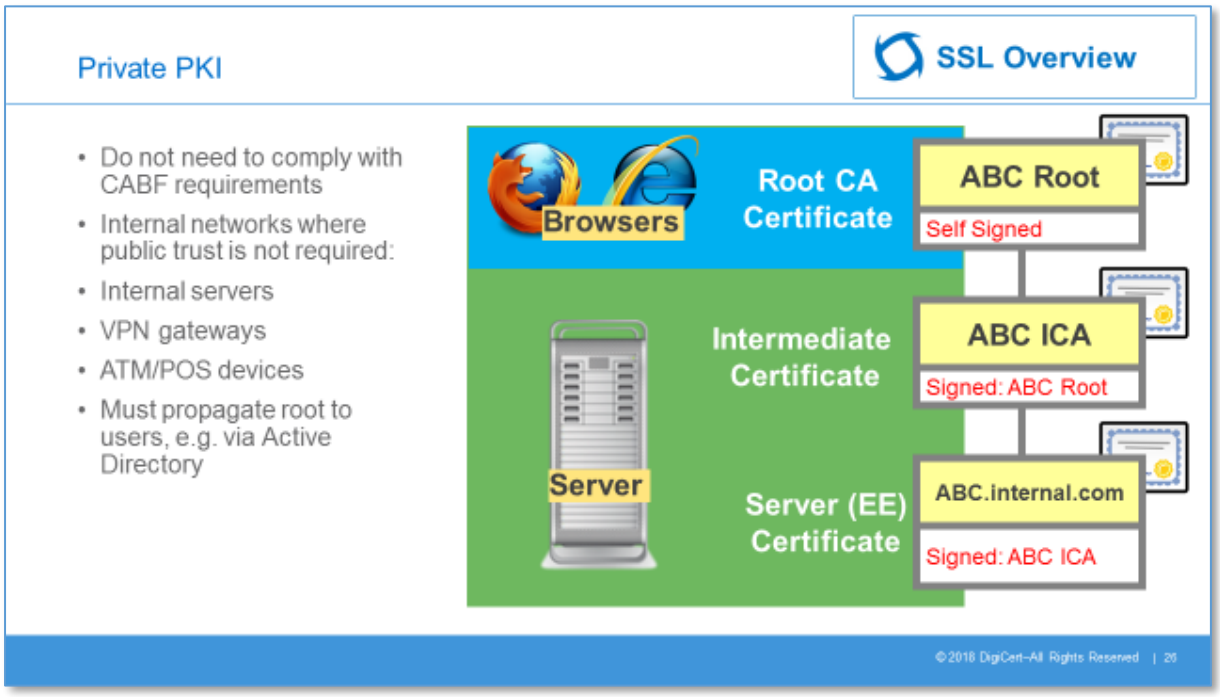


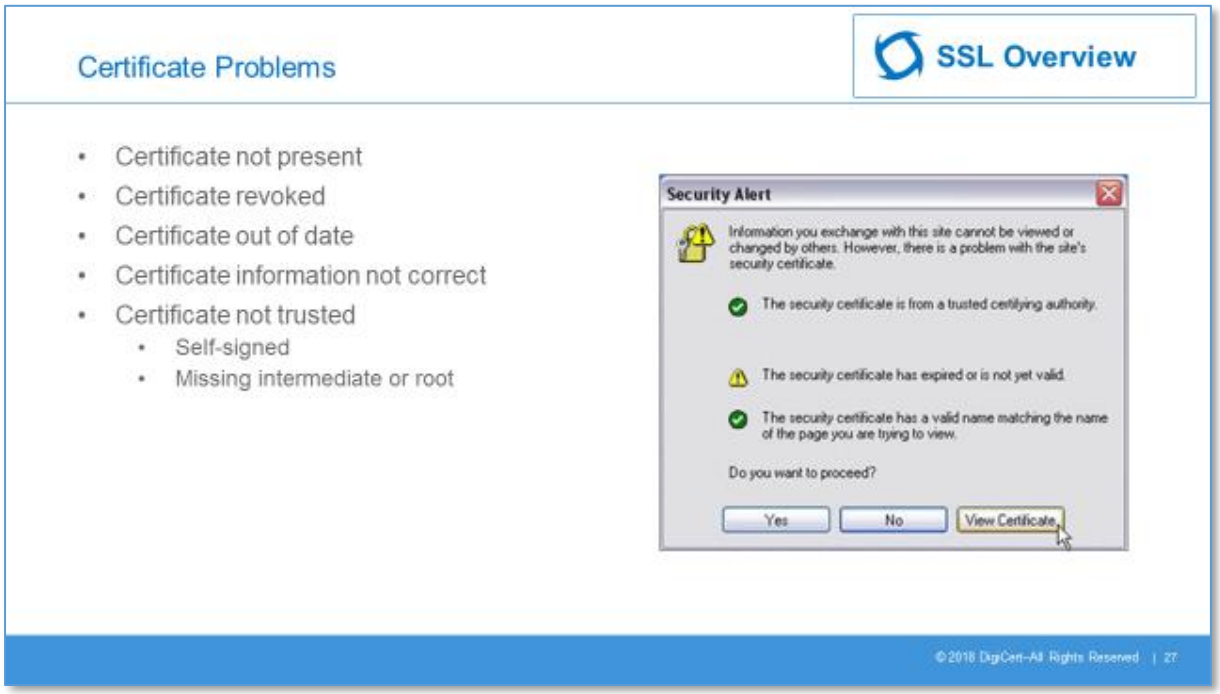
In some cases, web servers will send the client more than one intermediate certificate in case one of the intermediates certificates cannot be trusted because the client does not have the relevant root

certificate in their trust store. For example, Android mobile phones (pre v2.2) only contain a limited number of CA root certificates for VeriSign, Thawte and GeoTrust.

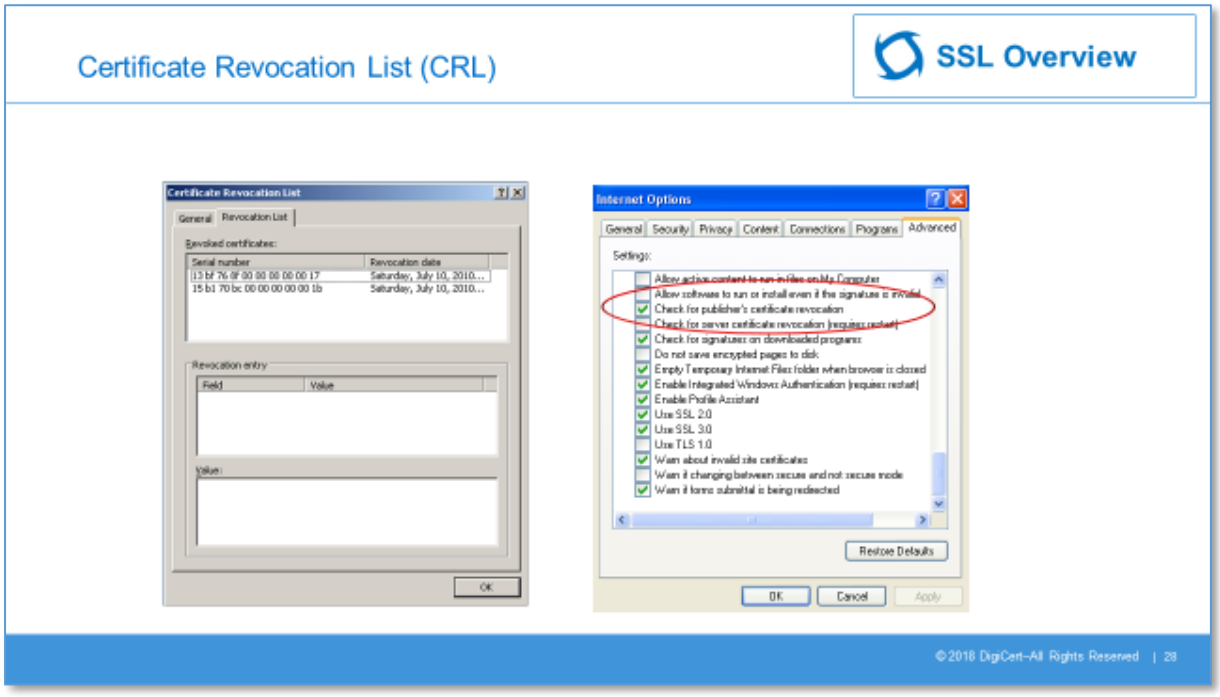
In the example above, the Google site sends the intermediate certificate (ICA) “Google Internet Authority G2” which is signed by “GeoTrust Global CA” root (in trust store). However – if for some reason “GeoTrust Global CA” is not in trust store, they also send ICA “GeoTrust Global CA” which is signed by “Equifax…” (in trust store).

Both “GeoTrust Global CA” certs contain the signature used to sign ICA “Google Internet Authority G2”, the difference is that one of them is self-signed and the other is signed by Equifax.



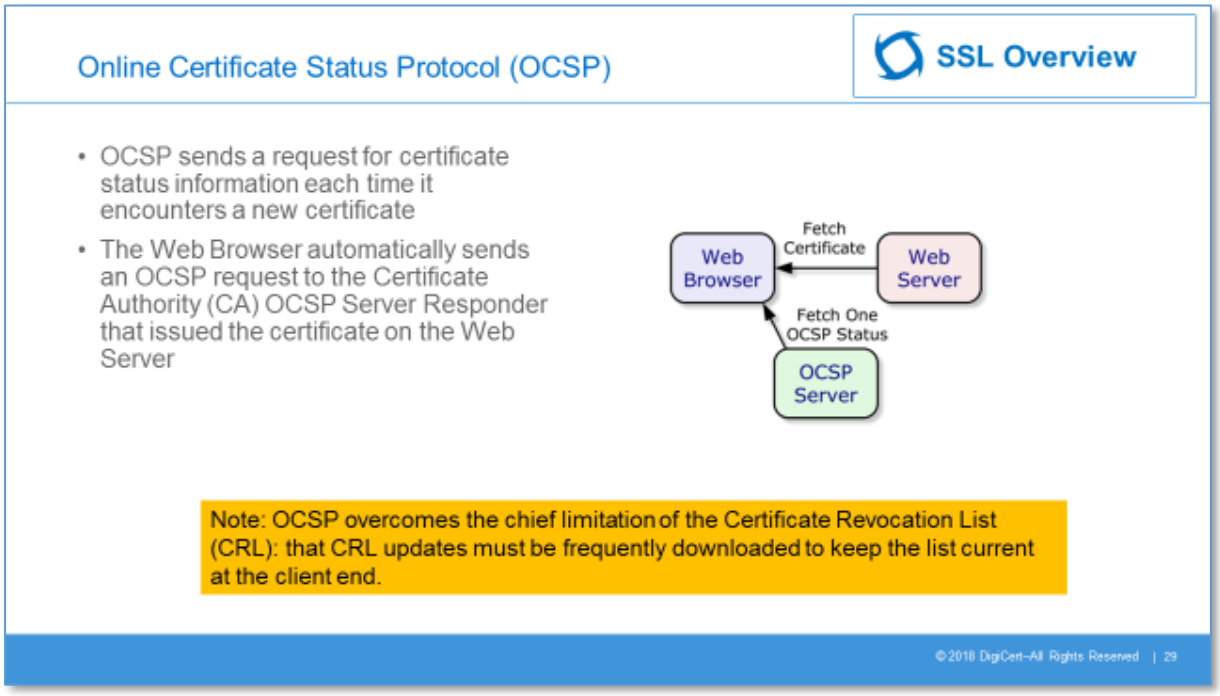


Certificate Revocation List (CRL)



The Certificate Revocation List is a list that contains all the serial numbers of certificates that have been revoked. These lists, however, need to be updated frequently by the certificate issuer. When the lists become outdated, they are no longer reliable for identifying revoked certificates. Keeping these lists continually updated is tedious, and the CRL process is often faulty due to the chance that revocation lists may not always be up-to-date.

Online Certificate Status Protocol (OCSP)



The Online Certificate Status Protocol (OCSP) is the fastest protocol we have for verifying certificate status. Here's how OCSP works: An end user sends a request to the server, requesting certificate

status information. Through the Online Certificate Status Protocol, a response is given as one of these four options “Success,” “Unauthorized,” “Malformed Request,” or “Try Later.” These responses indicate the status of the certificate and allow users to verify the security of the sites they’re using.

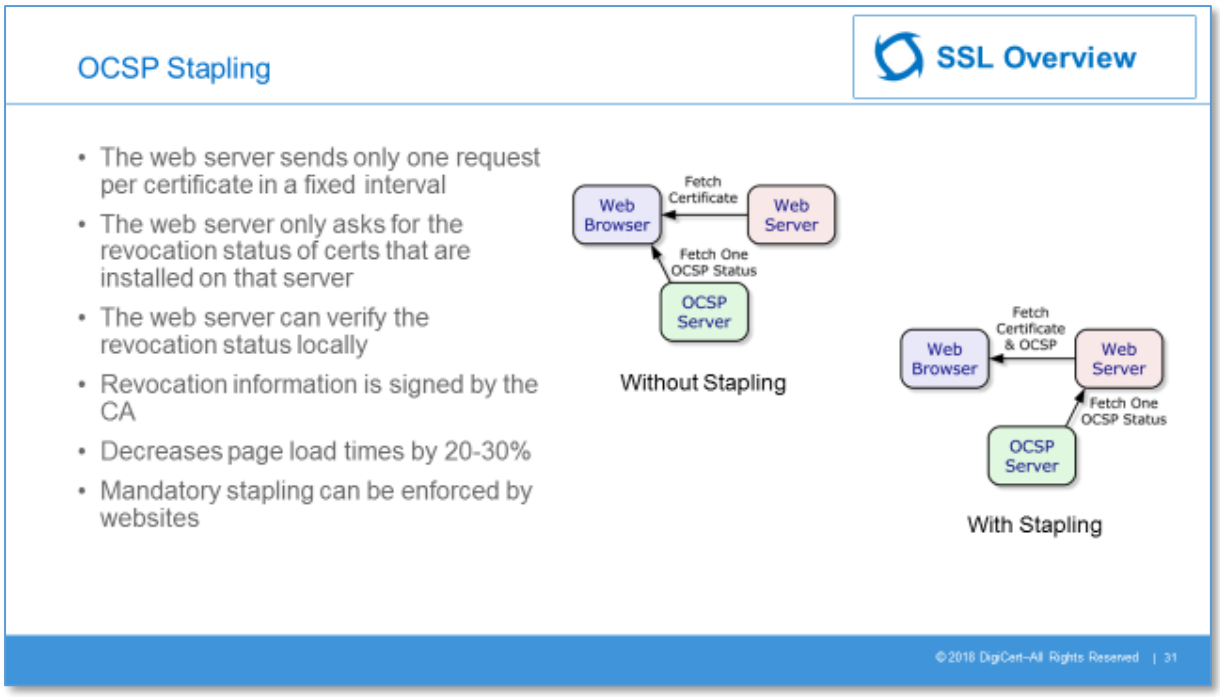
OCSP response times are in real-time. OSCP requests do not require the browser to check through long lists of revoked certificates to find certificate status. Likewise, OCSP requests contain much less information than CRL requests and can therefore be processed much quicker. This protocol dramatically streamlines the process of verifying a certificate. By quickening this process, OCSP has become the preferred protocol to obtaining the status of any certificate.

OCSP server uptime and response times should be a top priority in choosing a certificate issuer. End users should be cautious of companies who do not promote excellent server uptime and short OCSP responses. These metrics drastically affect site speed and page load time, which in turn affects the overall business. In one study, Amazon found that every 100 milliseconds of latency cost 1% in sales. In another study, Google found that just a 30-second delay for search results caused a 20% drop in traffic. The speed and delivery of any secure website is as integral to its success as the security itself. OCSP response times and uptime can make or break a website’s speed and certificate security.

However, browsers can't always communicate with the validation servers because of various technical problems and when something like this happens, the HTTPS connections should not be established; at least in theory.

However, because these failures can have a serious usability impact, browser vendors have decided ignore revocation checks that result in network errors. This is a referred to as a soft-fail.

After considering the drawbacks, Google decided to remove OCSP checks from future versions of Chrome and replace them with a local list of revoked certificates that can be updated without requiring a browser restart.



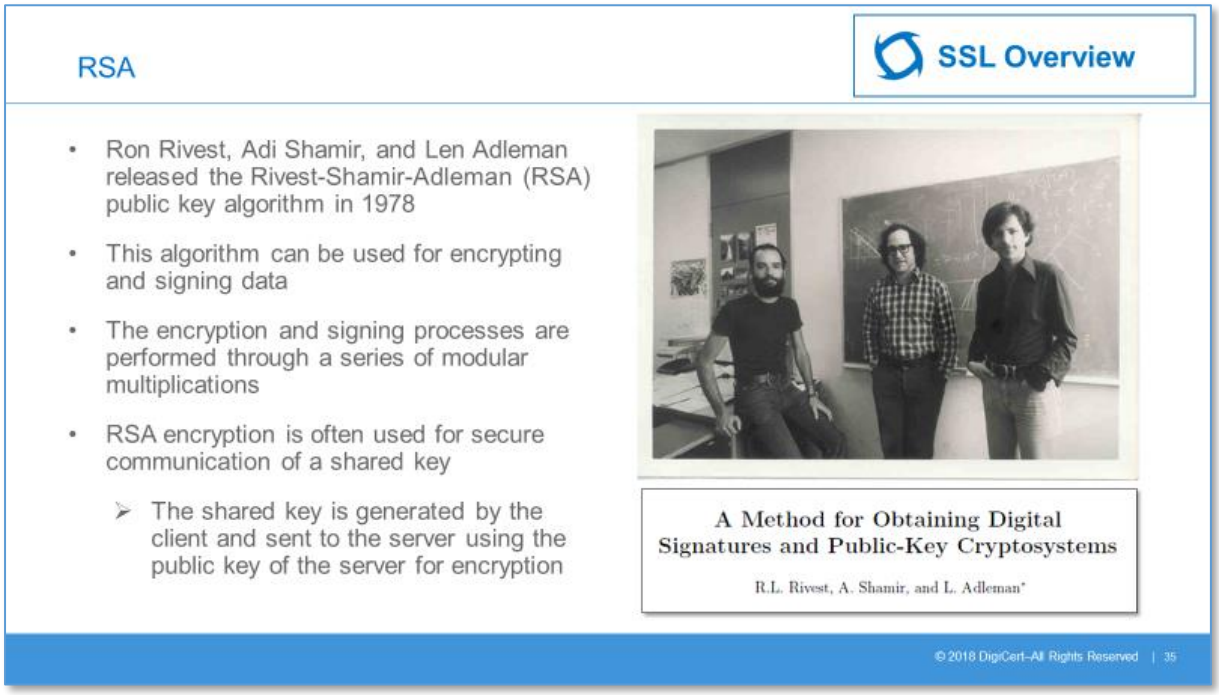
OCSP Stapling allows the Web Server to provide a time-stamped OCSP response, signed by the CA, to the initial handshake, eliminating the need for clients to contact the CA.

If you want a wider range of browsers to know if the certificate for your particular site is revoked, you can use a mechanism called ***must-staple*** (and a newer mechanism called ***expect-staple***) to indicate that it's mandatory to include a recent OCSP response along with the certificate itself. These mechanisms should be used with caution because, if you apply them and then don't set up the web server correctly, visitors can be locked out of visiting the site.

Must-staple is simply a flag in the certificate that puts a mandatory requirement on OSCP stapling presence and instructs the browser that the certificate must be served with a valid OCSP response or the browser should hard fail on the connection.

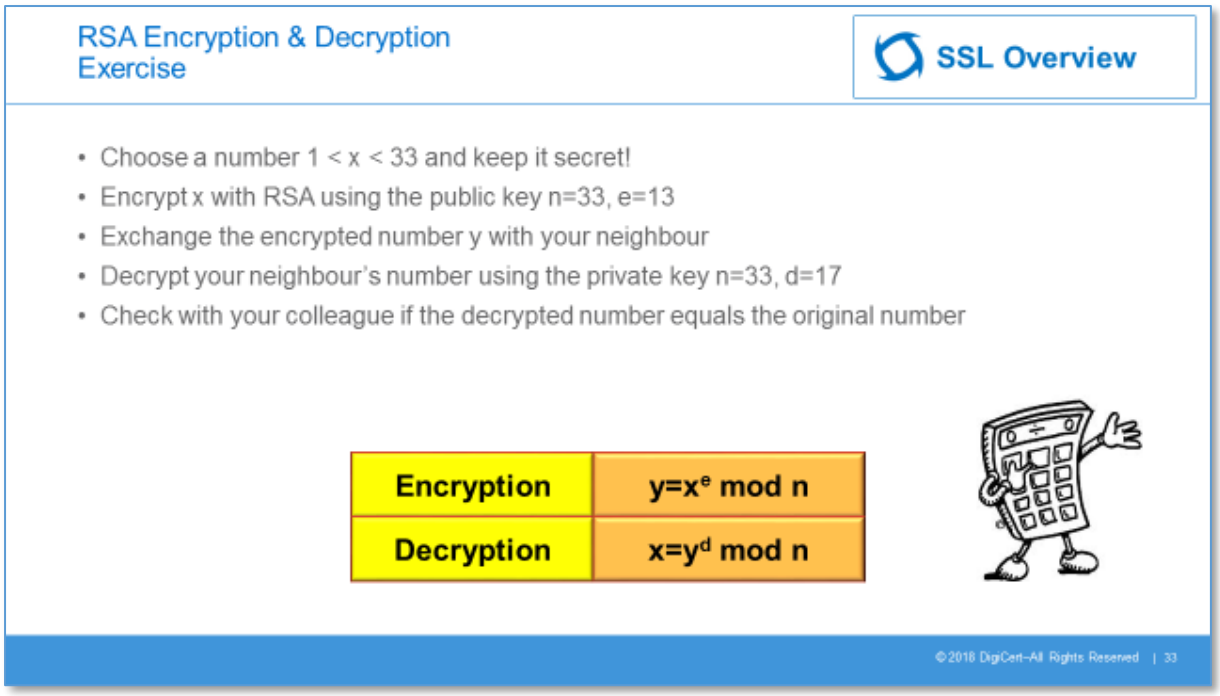
SSL Protocols & Algorithms

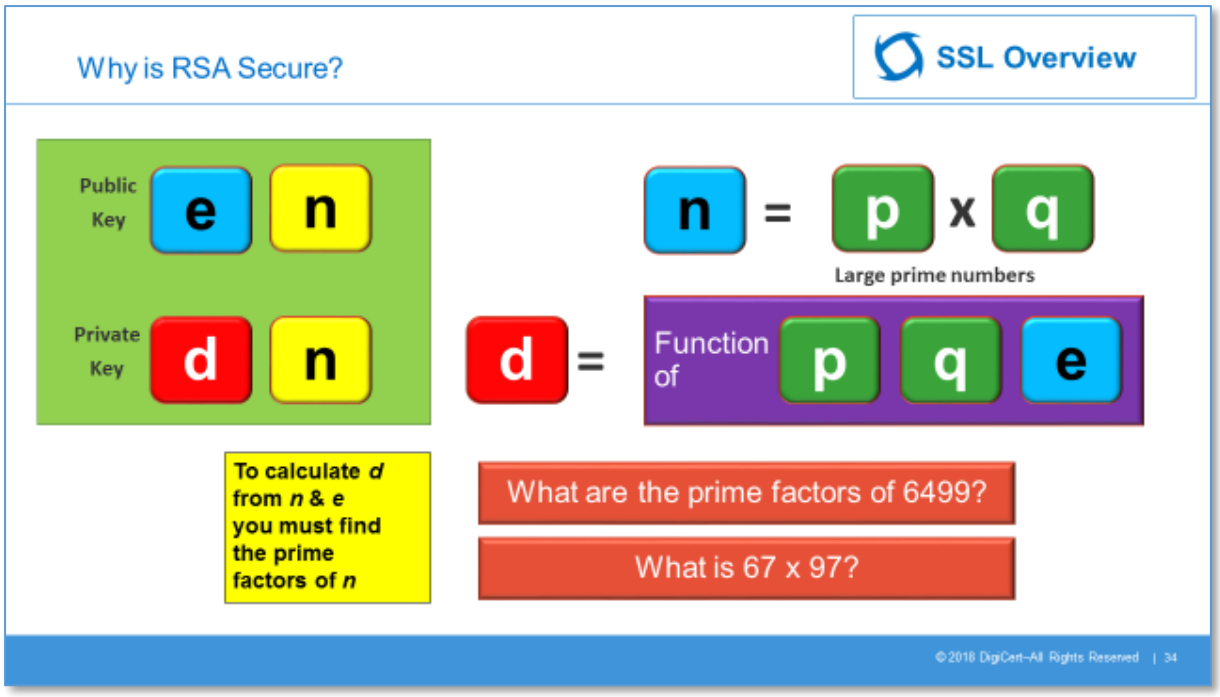
RSA



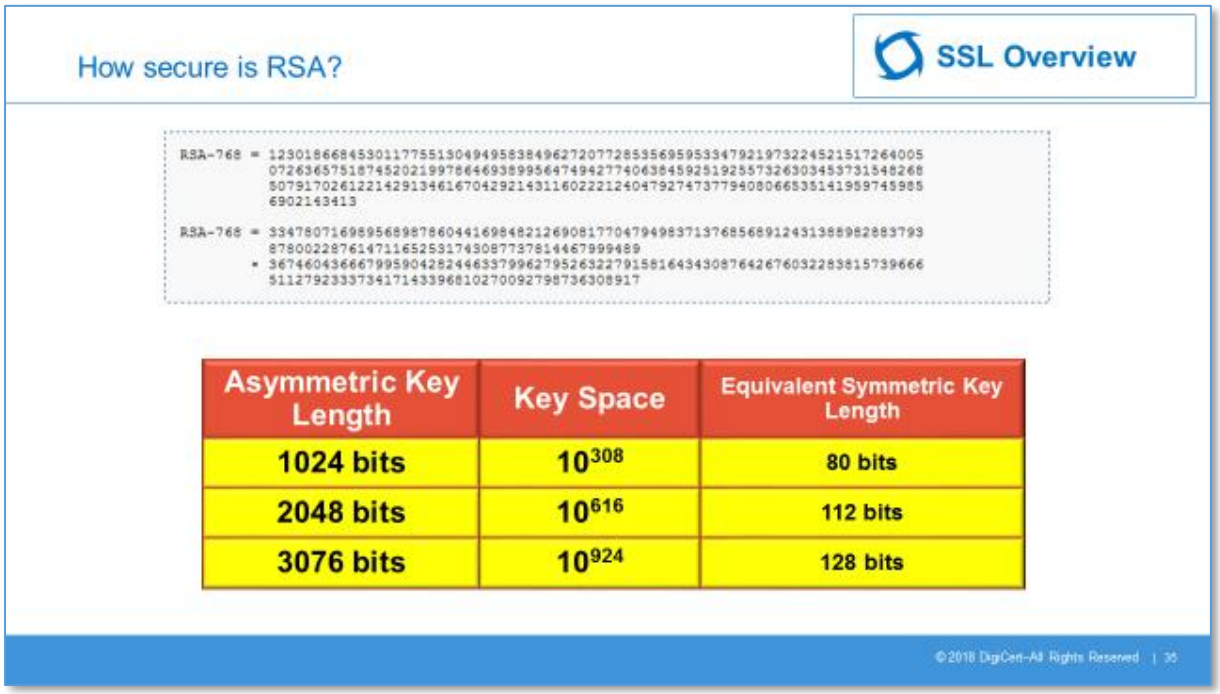
RSA stands for Rivest, Shamir and Adleman, who first publicly described the algorithm in 1977. For many years, RSA was the de facto standard for SSL/TLS connections on the internet.

RSA is typically used for authentication of the server and encryption of the key used by the chosen bulk encryption method.





RSA's main security foundation relies upon the fact that given two large prime numbers, a composite number (in this case n) can very easily be deduced by multiplying the two primes together. But, given just n, there is no known algorithm to efficiently determining n's prime factors.



Asymmetric Key Length	Key Space	Equivalent Symmetric Key Length
1024 bits	10^{308}	80 bits
2048 bits	10^{616}	112 bits
3076 bits	10^{924}	128 bits

In 2009, a 232 decimal digit number (RSA 768 - shown in the slide) was factored – but using 100s of computers over 2 years! A 1024-bit asymmetric key has a key space of 10^308 – 10^76 times more difficult.

Remember that symmetric keys are attacked by “brute force” – trying all possible keys. Finding prime factors is much easier, so asymmetric keys must be much longer for an equivalent level of

security – see table. Asymmetric keys up to 3k bits are required to keep pace with advances in Symmetric key lengths.

Diffie-Hellman

Diffie-Hellman Key Agreement

SSL Overview

- The Diffie-Hellman algorithm was invented in 1976 by Whitfield Diffie and Martin Hellman (US patent 4,200,770).
- The Diffie-Hellman algorithm is not for encryption or decryption but it enables two parties who are involved in communication to generate a shared secret key for exchanging information confidentially.
- They can do this even when an eavesdropper listens in on their entire conversation.

Whitfield 'Whit' Diffie

Martin Hellman

IEEE TRANSACTIONS ON INFORMATION THEORY, VOL. IT-22, NO. 6, NOVEMBER 1976

New Directions in Cryptography

Invited Paper

WHITFIELD DIFFIE AND MARTIN E. HELLMAN, MEMBER, IEEE

© 2018 DigiCert–All Rights Reserved | 39

The Diffie-Hellman algorithm is an alternative to RSA but is solely used for key agreement, i.e. securely agreeing the session key for bulk encryption.

Diffie-Hellman Key Exchange Exercise

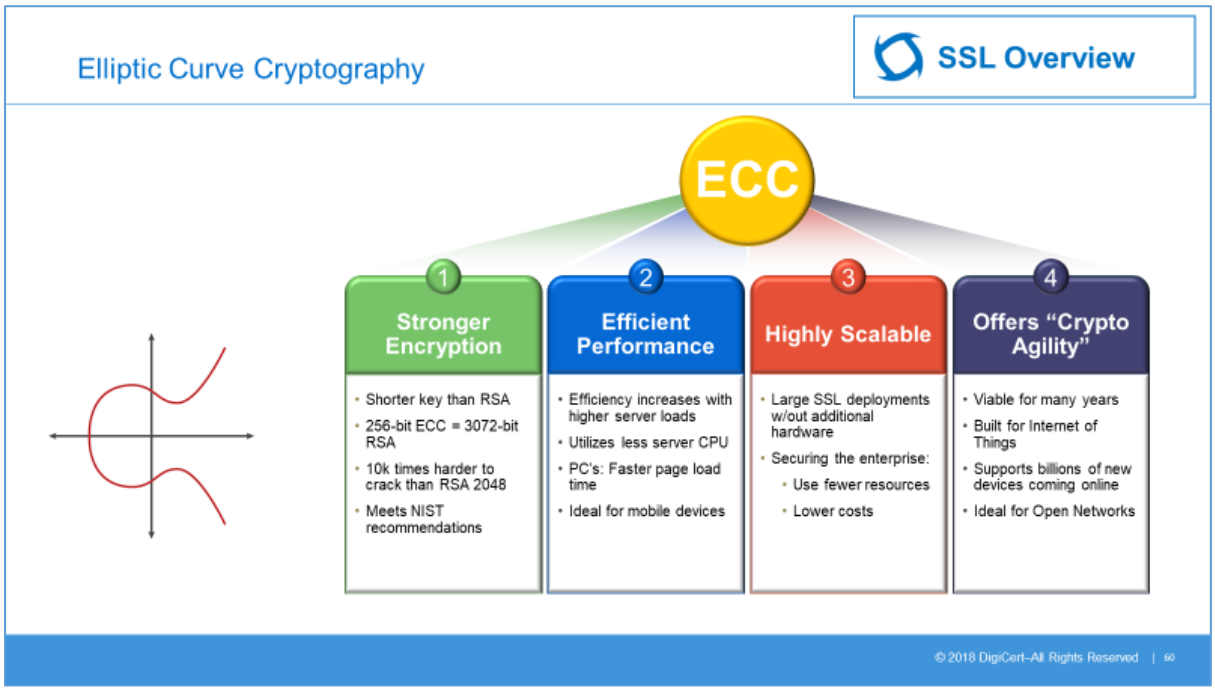
SSL Overview

- Choose a number 1 < x < 20 and keep it secret!
- Encrypt x with Diffie-Hellman using the public parameters n=31, g=3
- Exchange the encrypted number y with your neighbour
- Calculate the shared key (S) using your neighbour's encrypted number (y) and your number (x)
- Check with your neighbour if your values for S are the same

Encryption	**$y=g^x \bmod n$**
Shared Key	**$S=y^x \bmod n$**

© 2018 DigiCert–All Rights Reserved | 37

Elliptic Curve Cryptography



Elliptic-curve cryptography (ECC) is an approach to public-key cryptography based on the algebraic structure of elliptic curves over finite fields. ECC requires smaller keys compared to non-ECC cryptography to provide equivalent security.

Elliptic curves are applicable for key agreement, digital signatures, pseudo-random generators and other tasks. Indirectly, they can be used for encryption by combining the key agreement with a symmetric encryption scheme.

The primary benefit promised by elliptic curve cryptography is a smaller key size, reducing storage and transmission requirements, i.e. that an elliptic curve group could provide the same level of security afforded by an RSA-based system with a large modulus and correspondingly larger key: for example, a 256-bit elliptic curve public key should provide comparable security to a 3072-bit RSA public key.

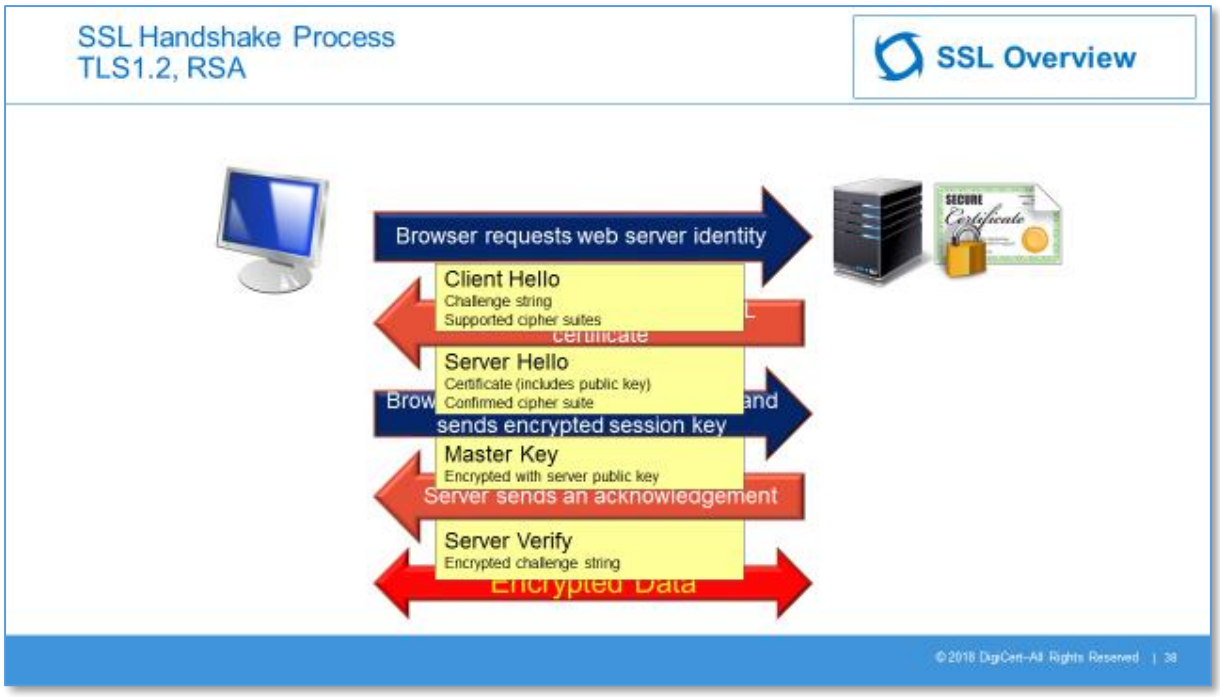
The U.S. National Institute of Standards and Technology (NIST) has endorsed elliptic curve cryptography in its Suite B set of recommended algorithms, specifically elliptic curve Diffie–Hellman (ECDH) for key exchange and Elliptic Curve Digital Signature Algorithm (ECDSA) for digital signature. The U.S. National Security Agency (NSA) allows their use for protecting information classified up to top secret with 384-bit keys. However, in August 2015, the NSA announced that it plans to replace Suite B with a new cipher suite due to concerns about quantum computing attacks on ECC.

SSL Handshake Details

SSL uses a series of client/server handshakes to enable a secure session. The two major phases of the SSL protocol are:

1. Establish private communications, and
2. Perform client authentication.

Note: In typical end-user/browser usage, SSL authentication is unilateral: only the server is authenticated (the client knows the server's identity), but not vice versa (the client remains unauthenticated or anonymous).



RSA

Message 1: Client Hello

This first step occurs when the client application tries to connect to a secure page. The application sends a random challenge string to the server and a list of cipher suites available. An example of a cipher suite is TLS_RSA_WITH_DES_CBC_SHA, where TLS is the protocol version, RSA is the algorithm that will be used for the key exchange, DES_CBC is the encryption algorithm (using a 56-bit key in CBC mode), and SHA is the hash function.

Message 2: Server Hello

The server asserts its identity by returning its secure server certificate.

The server will choose the strongest cipher that both the client and server support and confirm in the server "hello" message. (If there are no cipher suites that both parties support, the session is ended with a "handshake failure" alert.)

Message 3: Client Master Key

The client application verifies the server certificate by comparing the signature of the certification authority (CA) in the server's certificate to the public key of the CA embedded in the client application. If the client does not have a CA key, or the client CA certificate does not match the server CA, the user receives a message warning that this server contains a certificate not known by the client application. The user is given the opportunity to cancel the session, always trust the new CA certificate, or trust the CA certificate only for the current session.

After verifying the server, the client generates a master session key. This master session key is used as the seed to generate the client and server communications keys. Two symmetric key-pair sets are used: one for incoming communications and one for out going communications. Because the single master key was used as the seed, the server write-key equals the client read-key, and the server

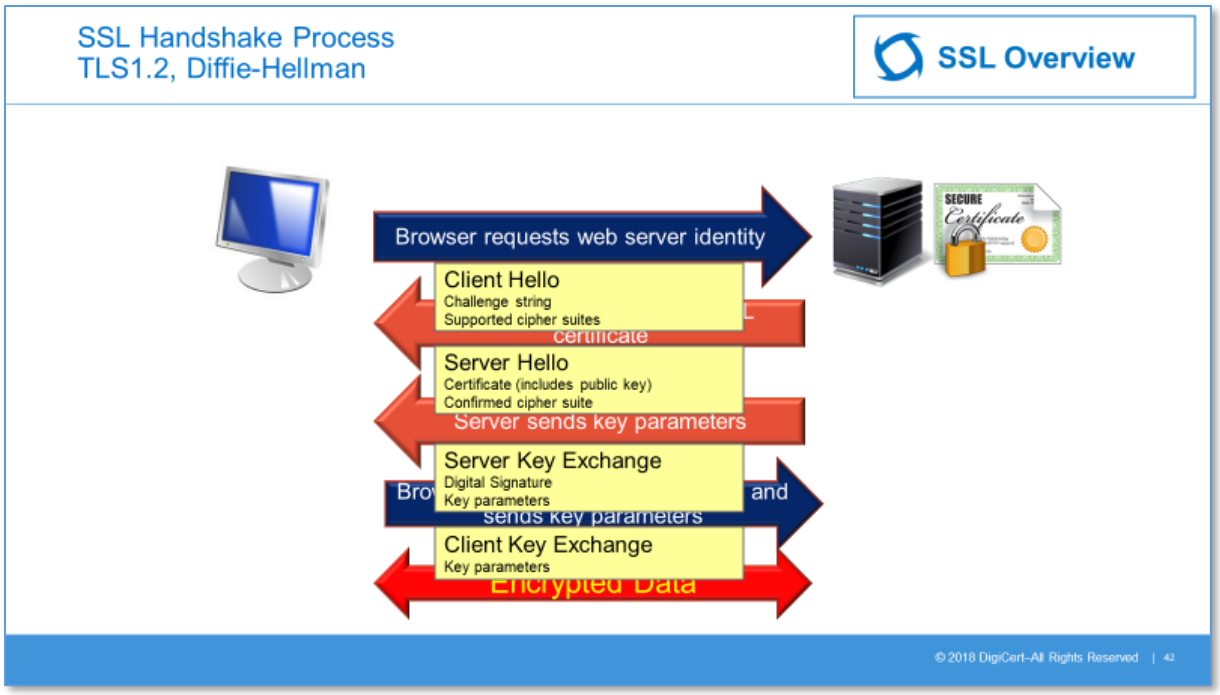
read-key equals the client write key. It is important that the master session key is generated by the client - not the server - thus providing another security layer to the user.

Finally, the client encrypts the session key with the server public key (contained in the server certificate) and sends it to the server.

Message 4: Server Verify

The server decrypts the master session key using the server private key and uses the session key to create the corresponding server key pairs. The server then returns the initial client challenge phrase, encrypted with the server-write key. This is confirmation of server authenticity, as only the master session key could have created the key used to encrypt the client challenge message.

At this point the server is authenticated, the communications protocols are set, and the (optional) client authentication phase begins.



Diffie-Hellman

Message 1: "Client Hello"

Just like in the RSA case, the client hello contains the protocol version, the client random, a list of cipher suites, and, optionally, the SNI extension. If the client speaks ECDHE, they include the list of curves they support.

Message 2: "Server Hello"

After receiving the client hello, the server picks the parameters for the handshake going forward, including the curve for ECDHE. The server "hello" message contains the server random, the server's chosen cipher suite, and the server's certificate.

The RSA and Diffie-Hellman handshakes start to differ at this point with a new message type.

Message 3: "Server Key Exchange"

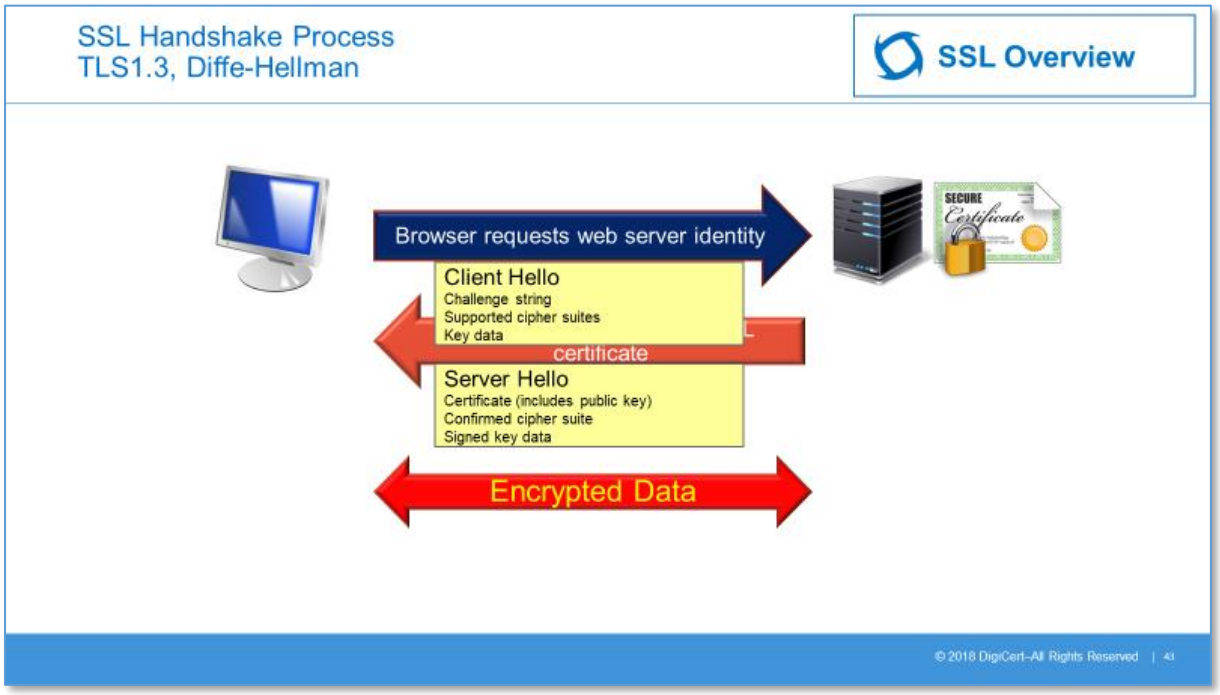
In order to start the Diffie-Hellman key exchange, the server needs to pick some starting parameters and send them to the client. The server also needs a way to prove that it has control of the private key, so the server computes a digital signature of all the messages up to this point. Both the Diffie-Hellman parameters and the signature are sent in this message.

Message 4: "Client Key Exchange"

After validating that the certificate is trusted and belongs to the site they are trying to reach, the client validates the digital signature sent from the server. They also send the client half of the Diffie-Hellman handshake.

At this point, both sides can compute the pre-master secret from the Diffie-Hellman parameters. With the pre-master secret and both client and server randoms, they can derive the same session key. They then exchange a short message to indicate that they the next message they send will be encrypted.

Just like in the RSA handshake, this handshake is officially complete when the client and server exchange "Finished" messages. Any subsequent communication between the two parties are encrypted with the session key.



TLS 1.3

Message 1: "Client Hello"

In TLS 1.3 a client starts by sending not only the Client Hello and the list of supported ciphers, but it also makes a guess as to which key agreement algorithm the server will choose and sends a key share for that.

That saves us a round trip, because as soon as the server selects the cipher suite and key agreement algorithm, it's ready to generate the key, as it already has the client key share. So it can switch to encrypted packets one whole round-trip in advance.

Message 2: "Server Hello"

The server sends the Server Hello, its key share, the certificate (now encrypted, since it has a key!), and already the Finished message.

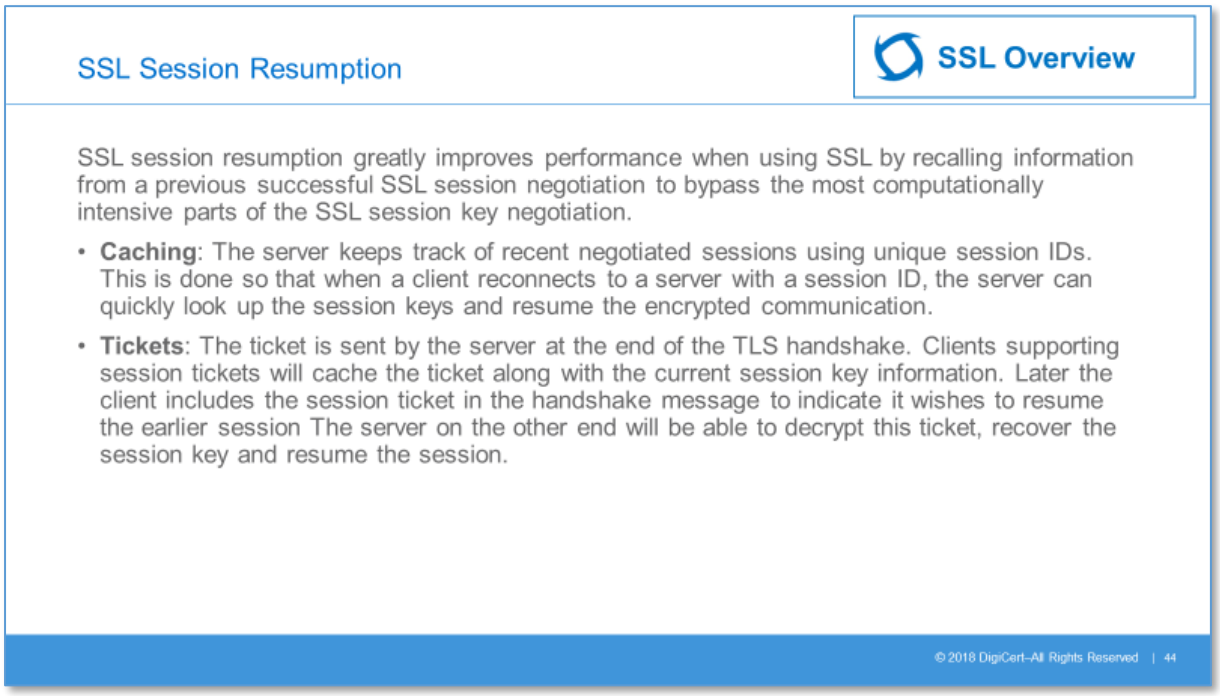
The client receives all that, generates the keys using the key share, checks the certificate and Finished, and it's immediately ready to send the HTTP request, after only one round-trip. Which can be hundreds of milliseconds.

Interoperability

All TLS versions and SSL 3.0 are very similar, and use compatible "Client Hello" messages; thus, supporting all of them is relatively easy. Similarly, servers can easily handle clients trying to use future versions of TLS as long as the "Client Hello" format remains compatible, and the client support the highest protocol version available in the server.

A TLS 1.2 client who wishes to negotiate with such older servers will send a normal TLS 1.2 "Client Hello", containing "TLS 1.2" in *ClientHello.client_version*. If the server does not support this version, it will respond with "Server Hello" containing an older version number. If the client agrees to use this version, the negotiation will proceed as appropriate for the negotiated protocol.

Session Resumption

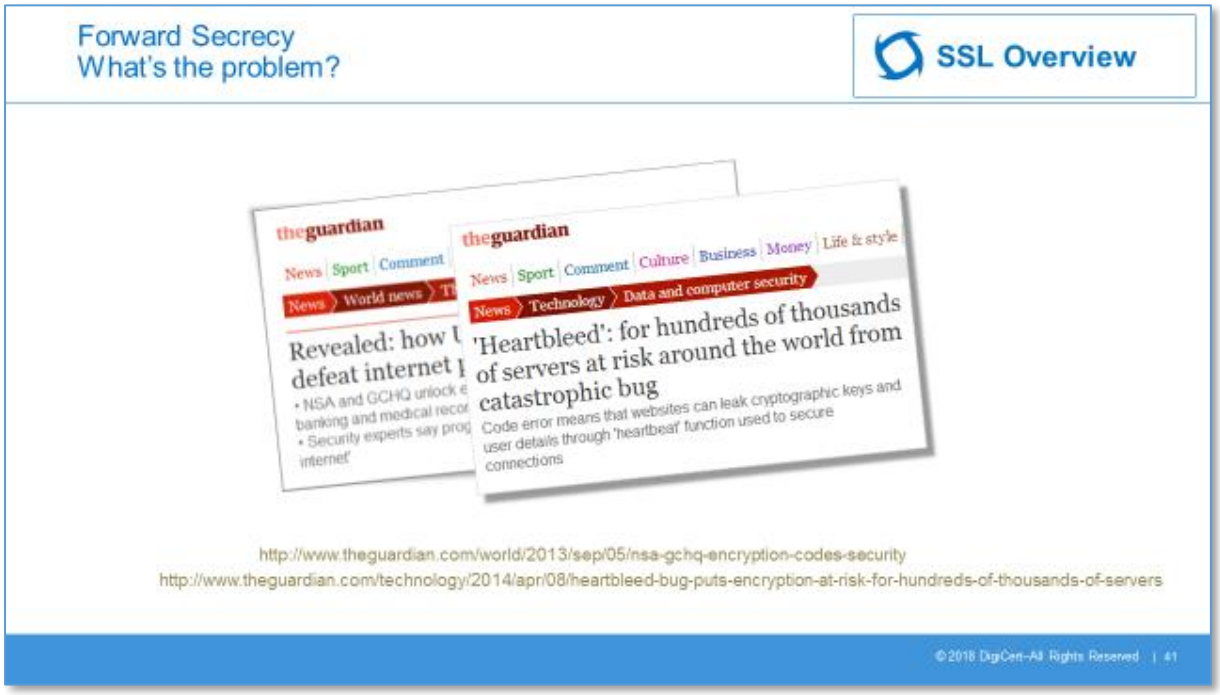


SSL session resumption greatly improves performance when using SSL by recalling information from a previous successful SSL session negotiation to bypass the most computationally intensive parts of the SSL session key negotiation. There are 2 approaches:

- Caching: Resuming an encrypted session through a session ID means that the server keeps track of recent negotiated sessions using unique session IDs. This is done so that when a client reconnects to a server with a session ID, the server can quickly look up the session keys and resume the encrypted communication.
- Tickets: Session resumption with session IDs has a major limitation: servers are responsible for remembering negotiated TLS sessions for a given period of time. It poses scalability issues for servers with a large load of concurrent connections per second and for servers

that want to cache sessions for a long time. Session ticket resumption is designed to address this issue. The idea is simple: outsource session storage to clients. A session ticket is a blob of a session key and associated information encrypted by a key which is only known by the server. The ticket is sent by the server at the end of the TLS handshake. Clients supporting session tickets will cache the ticket along with the current session key information. Later the client includes the session ticket in the handshake message to indicate it wishes to resume the earlier session. The server on the other end will be able to decrypt this ticket, recover the session key and resume the session.

Forward Secrecy



Many people have raised concerns that an adversary could record encrypted data and decrypt it later if they gain access to the relevant private key(s).

In particular, there are fears that government agencies have been recording encrypted data and may also have obtained some private keys.

Why is Forward Secrecy important?

SSL Overview

- Using typical HTTPS implementations based on RSA, an adversary could record encrypted data and decrypt it later if they gain access to the relevant private key(s)
- Using Forward Secrecy this approach is impossible

©2018 DigiCert-All Rights Reserved | 42

Prior to the implementation of Forward Secrecy, all data transmitted between a server and a client using RSA could be compromised if the server's private key was ever disclosed. In particular, an attacker could record encrypted traffic for any amount of time and store it until such a time that they had access to the private key. Once they have access to the private key, they can decrypt all historic data. This is possible because of the way that key material is exchanged between the client and the server when using RSA.

RSA key exchange

SSL Overview

- During the SSL handshake, the client receives the server's certificate
- This includes the server's public key
- Using RSA, the HTTPS session key is encrypted by the client using the server's public key and sent to the server across the network
- The server decrypts the session key using the corresponding private key

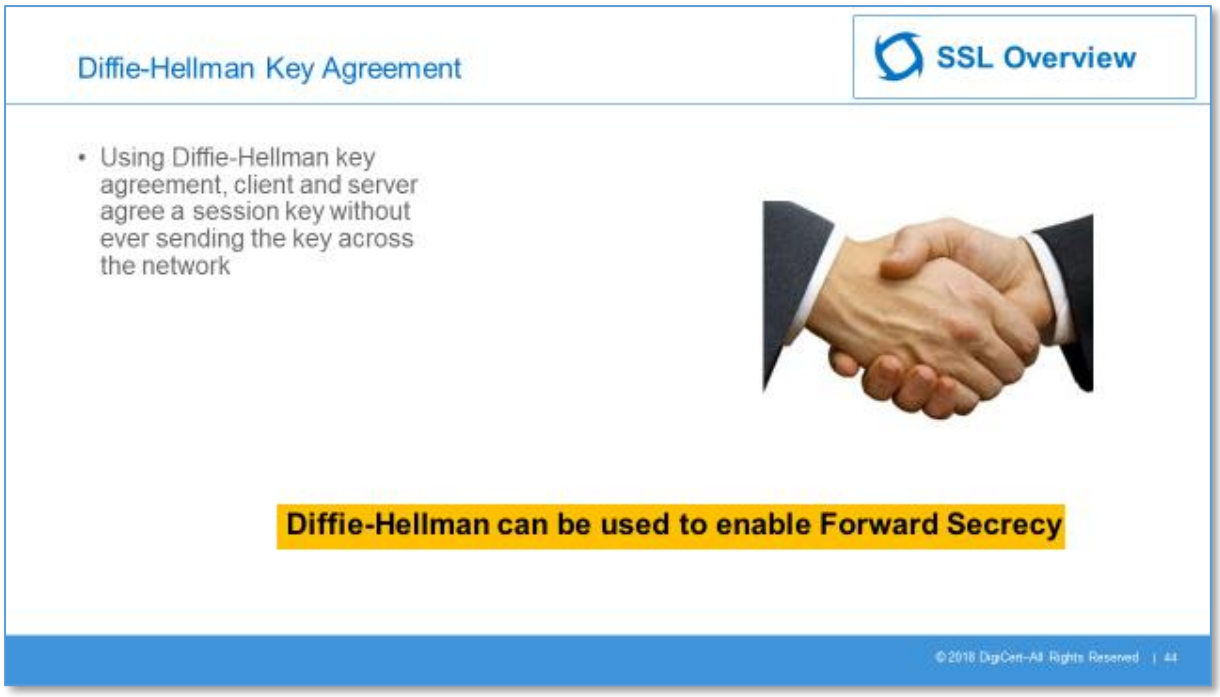
RSA is vulnerable if the private key is discovered

©2018 DigiCert-All Rights Reserved | 43

During the initial handshake, the client creates something called a Pre-Master Secret. The Pre-Master Secret is encrypted with the server's public key and sent to the server to protect it from being exposed whilst in transit. Once the server receives the PMS it can decrypt it with its own

private key and then both client and server have their own copy. From this, both client and server generate the symmetric session keys that will be used to exchange further data, the Master Secret.

Because the Pre-Master Secret is encrypted with the server's public key, exposure of the private key would allow an attacker to decrypt the data at any point in the future. This ability to decrypt historic data at any point represents quite a serious potential problem.



Forward secrecy is obtained by generating new key material for each session, that is generating an ephemeral key to be used for all messages of a conversation (e.g. by using Ephemeral Diffie–Hellman key exchange): in a worst-case scenario (such as arrest with live forensics performed on the device to retrieve the current ephemeral key in-memory), an adversary could only retroactively decode the ciphertext for the messages exchanged during that conversation, but none from the previous conversations.

What is Perfect Forward Secrecy?

SSL Overview

- Forward Secrecy uses Diffie-Hellman to generate a key for each session
- Perfect Forward Secrecy enhances Forward Secrecy by continuously changing the key material during each session
  - generating a new key for each message of a conversation

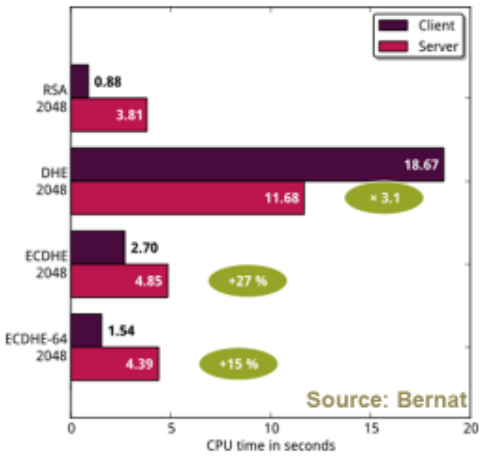
©2018 DigiCert–All Rights Reserved | 46

Perfect forward secrecy is obtained by continuously ratcheting the key material during each session, that is generating a new ephemeral key for each message of a conversation (e.g. by piggy-backing new Diffie–Hellman key exchanges on them): in the same worst-case scenario, an adversary could retroactively decode only the two last messages exchanged during that conversation, and still none from the previous conversations.

How do you implement Forward Secrecy?

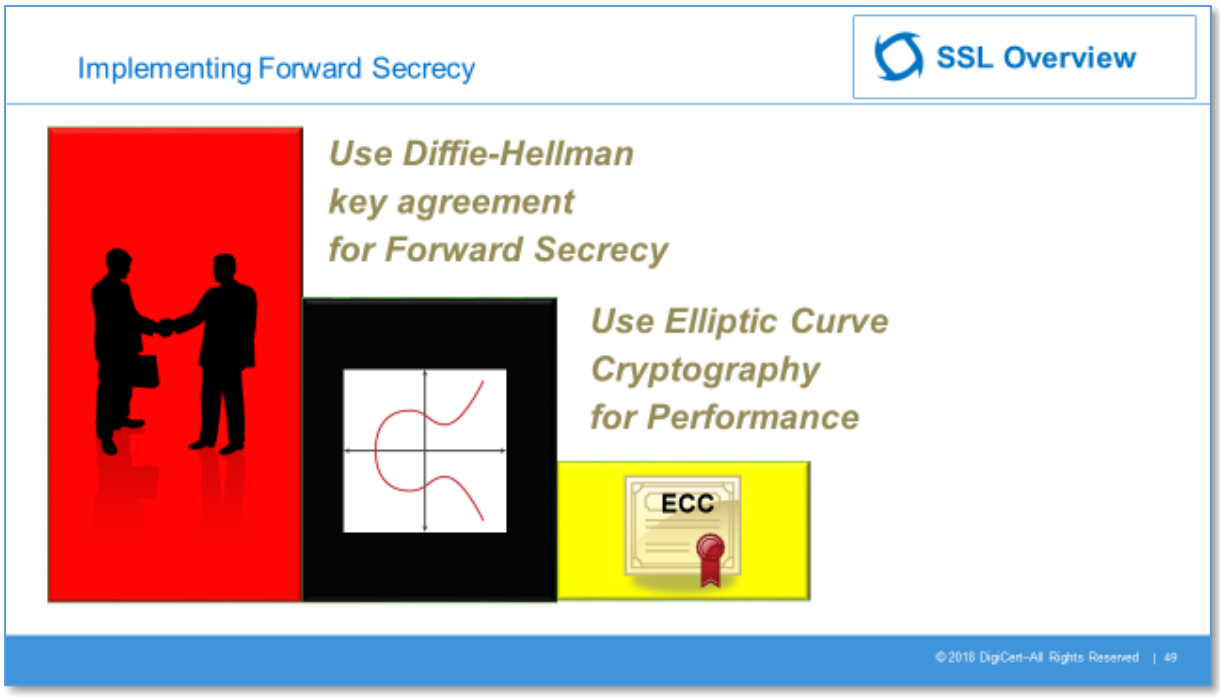
SSL Overview

- Diffie-Hellman Key Exchange (DHE) has a significant CPU overhead compared to RSA
- Elliptic Curve Diffie-Hellman (ECDHE) has minimal CPU overhead compared to RSA



©2018 DigiCert–All Rights Reserved | 47

Historically, Diffie-Hellman Key Exchange has been less popular than RSA because of its poor performance. However, using Elliptic Curve Diffie-Hellman gives very similar performance.



Therefore, the best way to implement Forward Secrecy is to use cipher suites which include Elliptic Curve Diffie-Hellman (ECDHE)

Cipher Suites

A cipher suite is a set of algorithms that help secure a network connection that uses Transport Layer Security (TLS) or Secure Socket Layer (SSL). Cipher suites are named combinations of:

- **Key Exchange** Algorithms (e.g. RSA, DH, ECDH, PSK). The key exchange algorithm is used to exchange a key between two devices. This key is used to encrypt and decrypt the messages being sent between two machines.
- **Authentication** (Signature) Algorithm (e.g. RSA, DSA) used to authenticate the server or client.
- **Bulk Encryption** Algorithms (e.g. AES, Camellia, ARIA). The bulk encryption algorithm is used to encrypt the data being sent.
- **Message Authentication Code** Algorithms (e.g. SHA-256). The MAC (hashing) algorithm provides data integrity checks to ensure that the data sent does not change in transit.

Here's an example of a cipher suite: **TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256**

- TLS is the protocol.
- ECDHE is the key exchange algorithm ephemeral Elliptic Curve Diffie Hellman (ECDHE).
- RSA is the authentication (signature) algorithm.
- AES_128_GCM is the bulk encryption algorithm.
- SHA-256 is the MAC algorithm.

Most browsers and servers have a list of cipher suites that they support, the two will compare the lists – in order of priority – against one another during the handshake in order to determine the security settings that will be used.

Algorithms supported in TLS 1.0-1.2 cipher suites

SSL Overview

Key exchange/ agreement	Authentication	Block/stream ciphers	Message authentication
RSA	RSA	RC4	Hash-based MD5
Diffie–Hellman	DSA	Triple DES	SHA hash function
ECDH	ECDSA	AES	
SRP		IDEA	
PSK		DES	
		Camellia	

https://en.wikipedia.org/wiki/Cipher_suite

©2018 DigiCert–All Rights Reserved | 50

Algorithms supported in TLS 1.3 cipher suites

SSL Overview

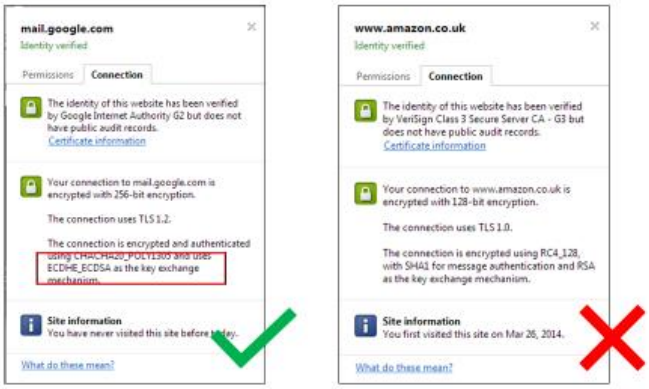
Key exchange/ agreement	Authentication	Block/stream ciphers	Message authentication
Diffie–Hellman	RSA	AES	SHA-2 hash function
ECDH	ECDSA	CHACHA20	
PSK			

©2018 DigiCert–All Rights Reserved | 51

Note that TLS 1.3 does not support RSA key exchange – so forward secrecy is supported by default.

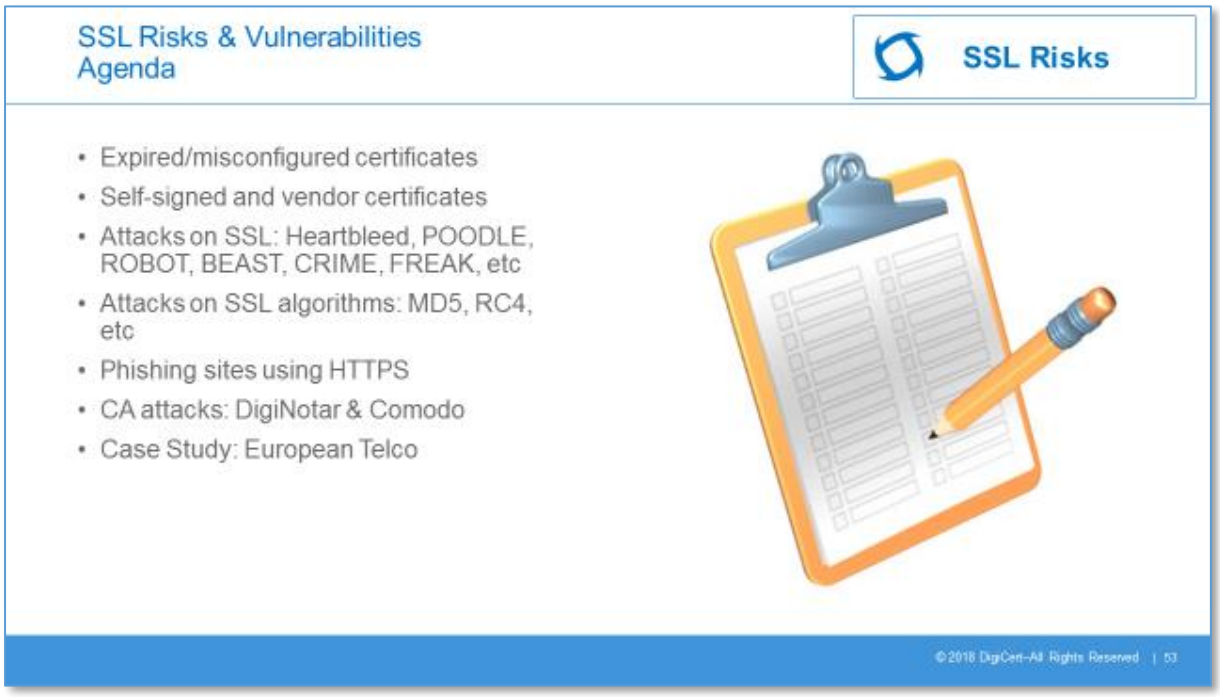
Are You Using Forward Secrecy?

SSL Overview



© 2018 DigiCert–All Rights Reserved | 64

SSL Risks & Vulnerabilities



Expired/misconfigured Certificates

Expired and misconfigure certificates can lead to extra costs, loss of revenue and serious brand damage and/or loss of reputation.

Expired Certificates

SSL Risks

Here Is The Ridiculous Reason 32 Million Telefonica (O2) Users Waved Goodbye To 4G Data Yesterday

Davey Winder Contributor
Cybersecurity
I report and analyse breaking cybersecurity and privacy stories

An initial root cause analysis indicates that the main issue was an expired certificate in the software versions installed with these customers. A complete and comprehensive root cause analysis is still in progress. Our focus is now on solving the immediate issues.

O2 suffered a massive outage after a minor mistake (AP Photo/Matthias Schrader) associated press

Yesterday was not a good day for users of the second biggest mobile network operator in the United Kingdom, Telefonica U.K. which is better known as O2, as its customers lost access to the 4G network. Nor was it a good day for Telefonica itself which also owns GiffGaff and provides virtual network access to users of Lyca Mobile, Sky Mobile and Tesco Mobile who also found themselves without 4G data. In total, 32 million users found themselves without data or SMS

© 2018 DigiCert–All Rights Reserved | 57

December 6, 2018: 32 million O2 users found themselves without data or SMS message services from breakfast to bedtime. Users of SoftBank mobile services in Japan also found themselves caught up in the communications crisis. Now Ericsson, the Swedish cellular network infrastructure giant which provides the common link in all of this, has revealed the reason for the data blackout. It was, frankly speaking, a rather ridiculous one. Ericsson confirmed that "an initial root cause analysis indicates that the main issue was an expired certificate in the software versions installed with these customers."

CLOUD

Microsoft coughs up compensation for Azure cloud cock-up

Embarrassing SSL cert snafu probe launched

By Brid-Aine Parnell, 25th February 2013

SSL Risks

The Register®

18

RELATED STORIES

Updated
Microsoft SkyDrive, Outlook stricken by cloud outage

Azure is Microsoft's billion -dollar baby – maybe

Microsoft hosts bar mitzvah for mature Azure

Outages plague Hotmail and Outlook users

VPS.net cloud hosting has a hiccup

Choosing the right database for your social game

Microsoft has vowed to compensate users of its Azure cloud after an expired SSL certificate took the service offline.

Victims of the Blue Sky of Death, which lasted 12 hours, will get credits as per their service-level agreement (SLA), the Redmond giant confirmed on its website.

Windows Azure was knocked offline globally last night. Microsoft made the schoolboy error of forgetting to renew a security certificate. The online storage system collapsed, causing parts of the software titan's cloud to fail, eventually leading to Xbox Live going down.

Availability was mostly back by 1am PST on Saturday, although it wasn't until 8pm that Microsoft was able to confirm full availability worldwide.

"Given the scope of the outage, we will proactively provide credits to impacted customers in accordance with our SLA," wrote the general manager of Azure business and operations Steven Martin.

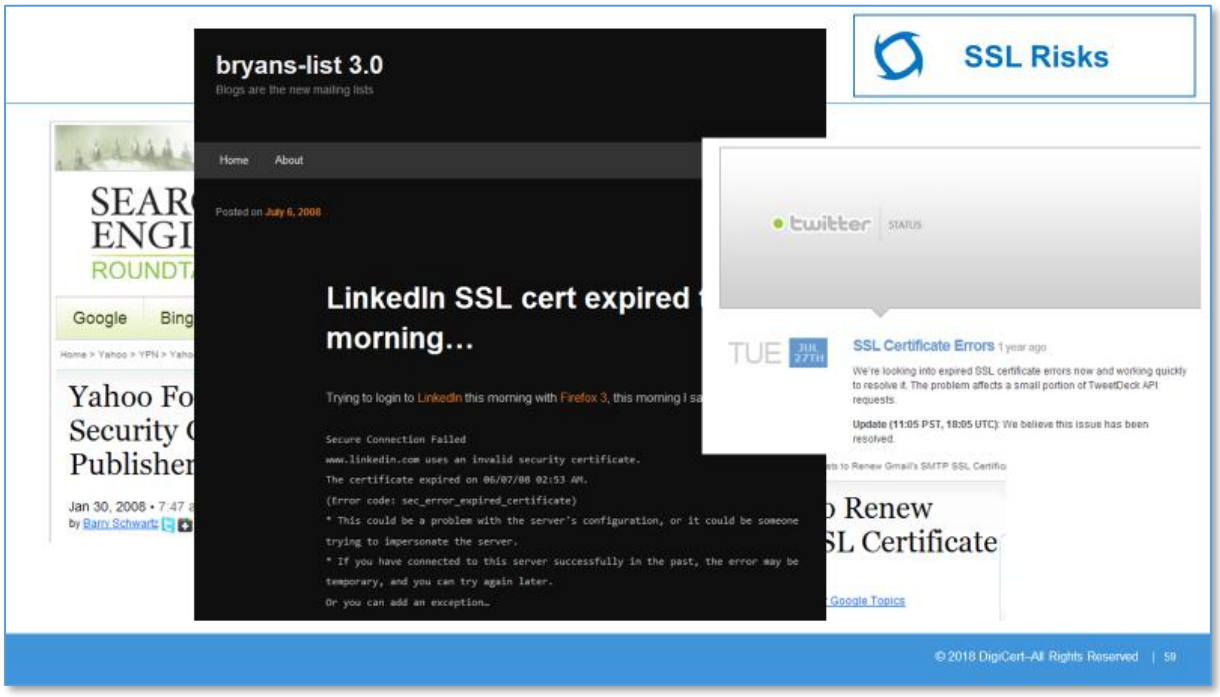
While compensation will be welcome, users will be more concerned with how Microsoft managed to trip up on such a basic error. Martin said that the company will do a full root cause analysis (RCA) to figure out how the SSL certificate expiry was allowed to happen.

GFI MailEssentials Online
Protect your business in minutes.
Get started now!
Find out more and try it for free
GFI

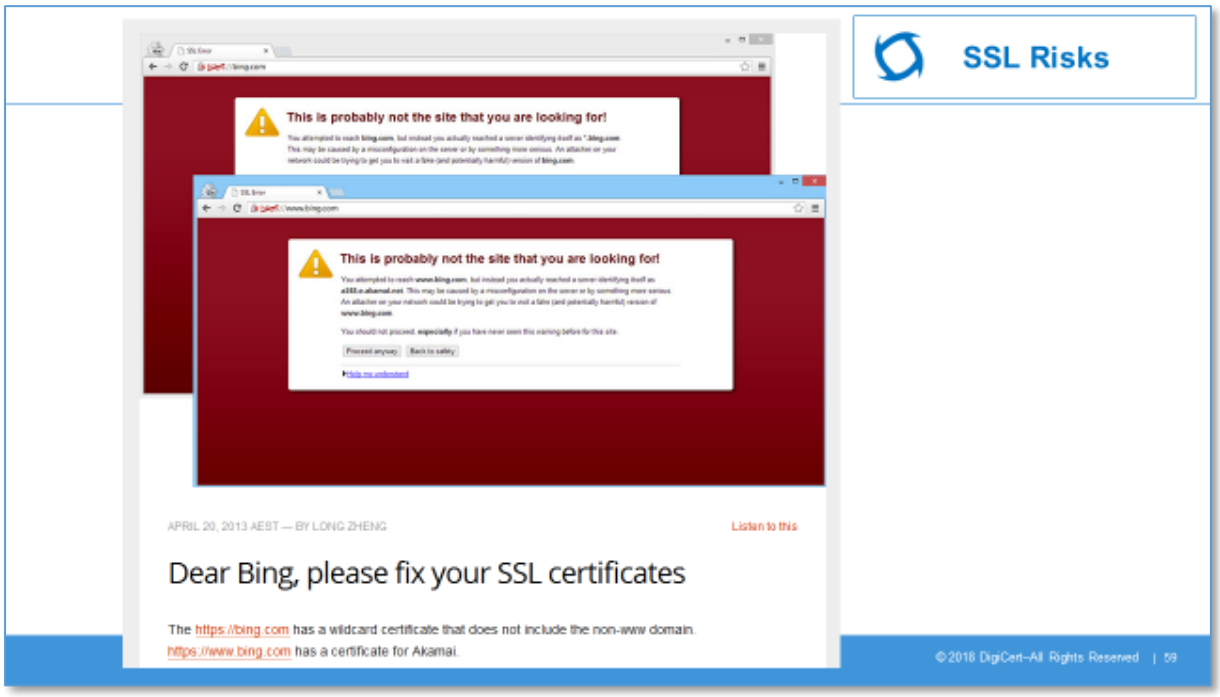
© 2018 DigiCert–All Rights Reserved | 58

This example is from 2013. Windows Azure was knocked offline globally for 12 hours because Microsoft made the “schoolboy error” of forgetting to renew a security certificate. The online

storage system collapsed, causing parts of Microsoft's cloud to fail, eventually leading to Xbox Live going down. Microsoft subsequently had to compensate users of its Azure cloud.

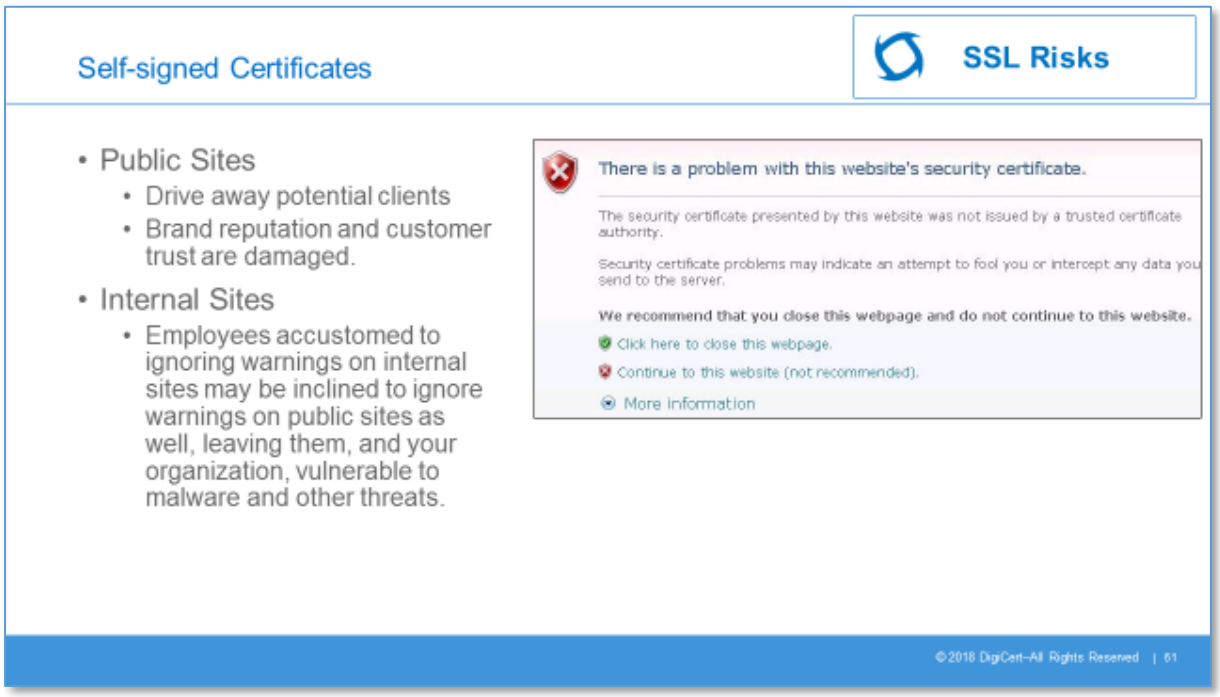


Apparently no-one is perfect when it comes to renewing SSL Certificates. Google forgot to renew an SSL Certificate for www.googleadservices.com resulting in an error displayed on users' computers for a large part of the day. It primarily affected sites using Google checkout or AdWords conversion tracking. Other examples include Yahoo, LinkedIn and Twitter.



Another example from Microsoft – showing a typical browser warning.

Self-signed & Vendor Certificates

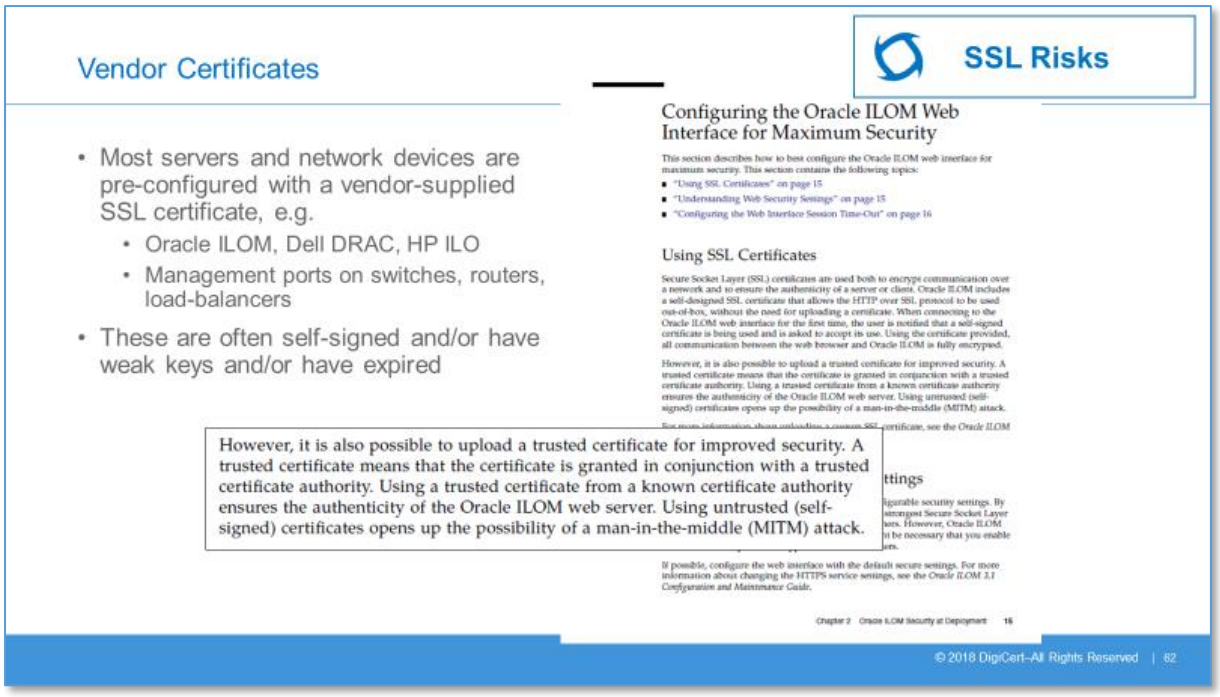


Many organizations are tempted to use self-signed SSL Certificates instead of those issued and verified by a trusted Certificate Authority mainly because of the price difference. Unlike CA issued certificates, self-signed certificates are free of charge. What most users are not aware of is that self-signed certificates can end up costing them more in the long run.

While self-signed SSL Certificates also encrypt customers' log-in and other personal account credentials, they prompt most web servers to display a security alert because the certificate was not verified by a trusted Certificate Authority. Often the alerts advise the visitor to abort browsing the page for security reasons.

On public sites, the security warnings associated with self-signed SSL Certificates drive away potential clients for fear that the website does not secure their credentials. Both brand reputation and customer trust are damaged.

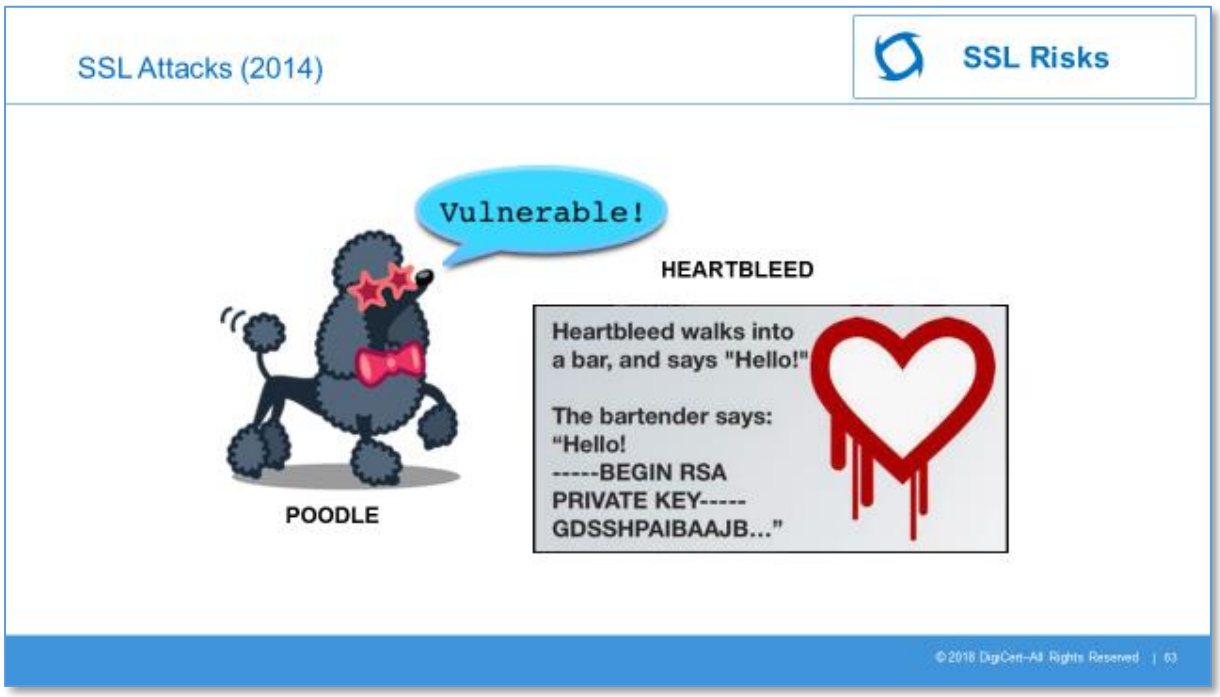
While the dangers of using self-signed certificates on public sites may be obvious, there is also risk to using them internally. Self-signed certificates on internal sites (e.g. employee portals) still result in browser warnings. Many organizations advise employees to simply ignore the warnings, since they know the internal site is safe, but this can encourage dangerous public browsing behaviour. Employees accustomed to ignoring warnings on internal sites may be inclined to ignore warnings on public sites as well, leaving them, and your organization, vulnerable to malware and other threats.



Integrated Lights-Out Management (ILO/ILOM) makes it possible to perform activities on a server from a remote location. The ILO card has a separate network connection (and its own IP address) to which one can connect via HTTPS. You can use ILO's digital certificate capabilities to prevent malicious attacks (such as Trojan horse attacks) where an impostor appears to be a trusted ILO web server. For example, if someone put a server that emulated ILO onto a corporate network, that server would not have a legitimate ILO certificate. If any user browsed to this emulated ILO device, the browser would flag the lack of a recognized certificate.

These ILO systems are usually configured with a default "vendor" certificates. Unfortunately, these are often self-signed and/or have weak keys and/or have expired already, which means that they will often generate browser warnings. It is strongly recommended to replace these with more secure SSL certificates.

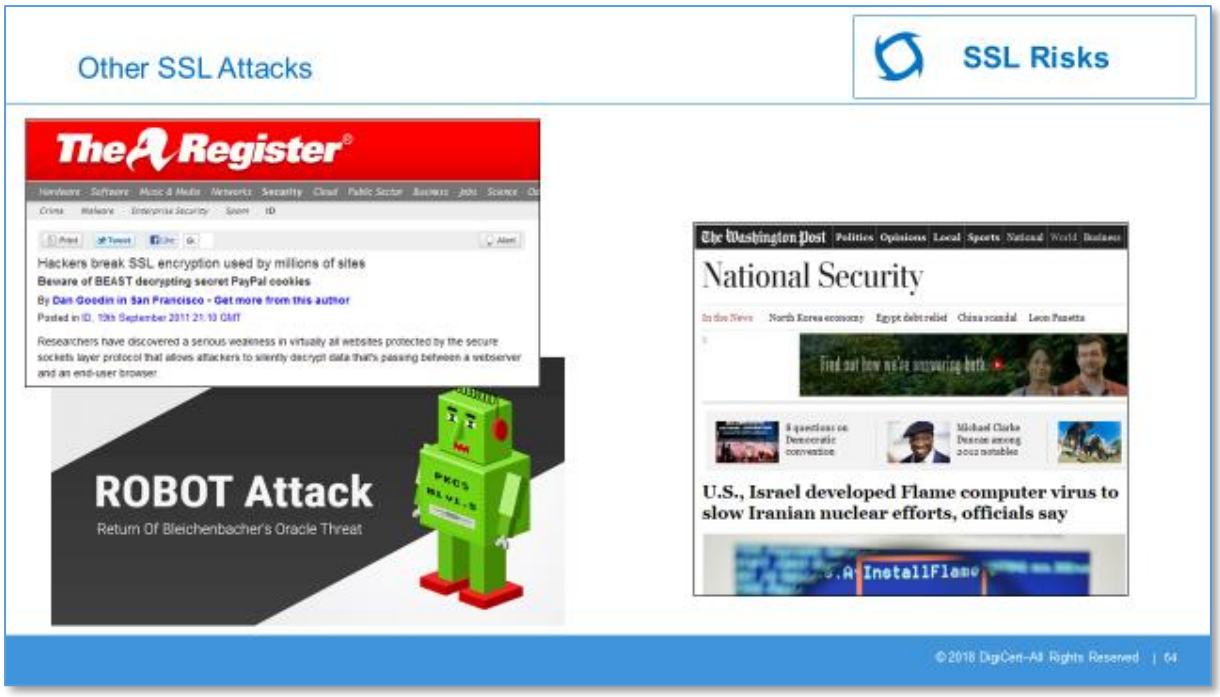
Attacks on SSL



Heartbleed was caused by a security bug in the OpenSSL cryptography library, which is a widely used implementation of the TLS protocol. It was introduced into the software in 2012 and publicly disclosed in April 2014. Heartbleed may be exploited regardless of whether the vulnerable OpenSSL instance is running as a TLS server or client. It results from improper input validation (due to a missing bounds check) in the implementation of the TLS heartbeat extension, thus the bug's name derives from heartbeat. The vulnerability is classified as a buffer over-read, a situation where more data can be read than should be allowed.

TLS implementations other than OpenSSL, such as GnuTLS, Mozilla's Network Security Services, and the Windows platform implementation of TLS, were not affected because the defect existed in the OpenSSL's implementation of TLS rather than in the protocol itself.

The **POODLE** attack (which stands for "Padding Oracle On Downgraded Legacy Encryption") is a man-in-the-middle exploit which takes advantage of Internet and security software clients' fallback to SSL 3.0. If attackers successfully exploit this vulnerability, on average, they only need to make 256 SSL 3.0 requests to reveal one byte of encrypted messages. Bodo Möller, Thai Duong and Krzysztof Kotowicz from the Google Security Team discovered this vulnerability; they disclosed the vulnerability publicly on October 14, 2014. On December 8, 2014 a variation of the POODLE vulnerability that affected TLS was announced.

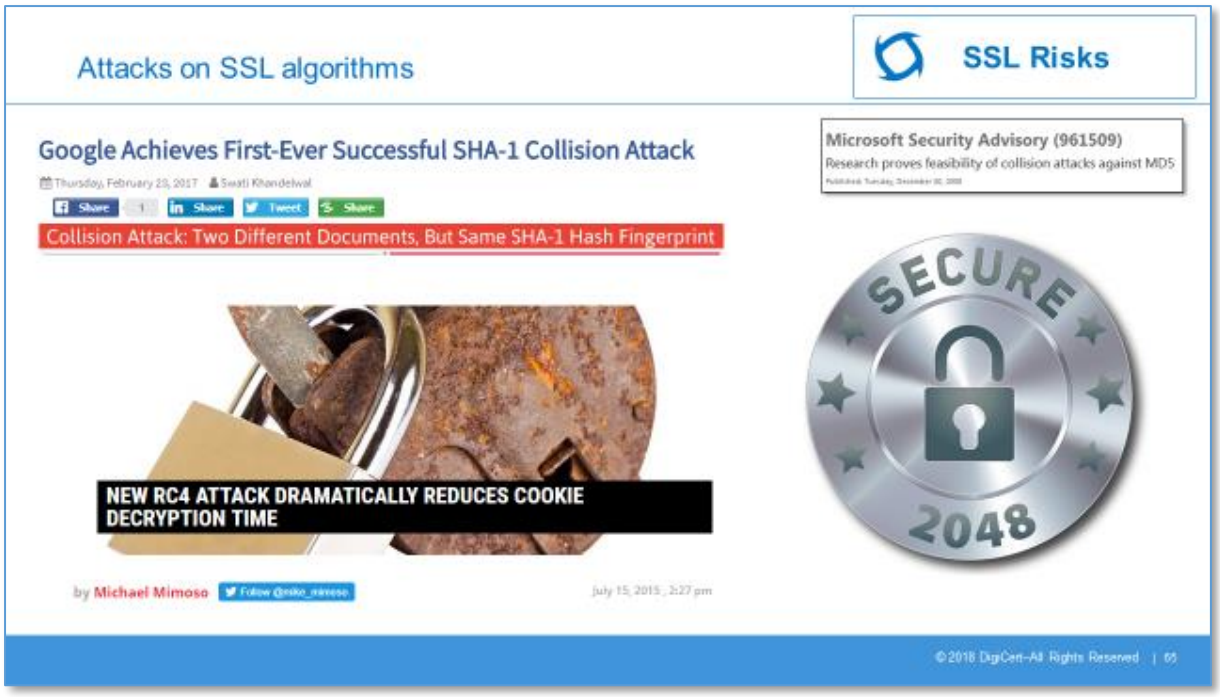


Flame was signed with a fraudulent certificate purportedly from the Microsoft Enforced Licensing Intermediate PCA certificate authority. The malware authors identified a Microsoft Terminal Server Licensing Service certificate that still used the weak MD5 hashing algorithm, and produced a counterfeit certificate that was used to sign some components of the malware to make them appear to have originated from Microsoft. (A successful collision attack against a certificate was previously demonstrated in 2008.)

Short for Browser Exploit Against SSL/TLS, **BEAST** is a browser exploit against SSL/TLS that was revealed in late September 2011. This attack leverages weaknesses in cipher block chaining (CBC) to exploit the SSL/TLS protocol. The CBC vulnerability can enable man-in-the-middle (MITM) attacks against SSL in order to silently decrypt and obtain authentication tokens, thereby providing hackers access to data passed between a Web server and the Web browser accessing the server. BEAST is purely a client-side vulnerability. Since it had been released to the public, most major browsers addressed it using a technique called 1/n-1 split.

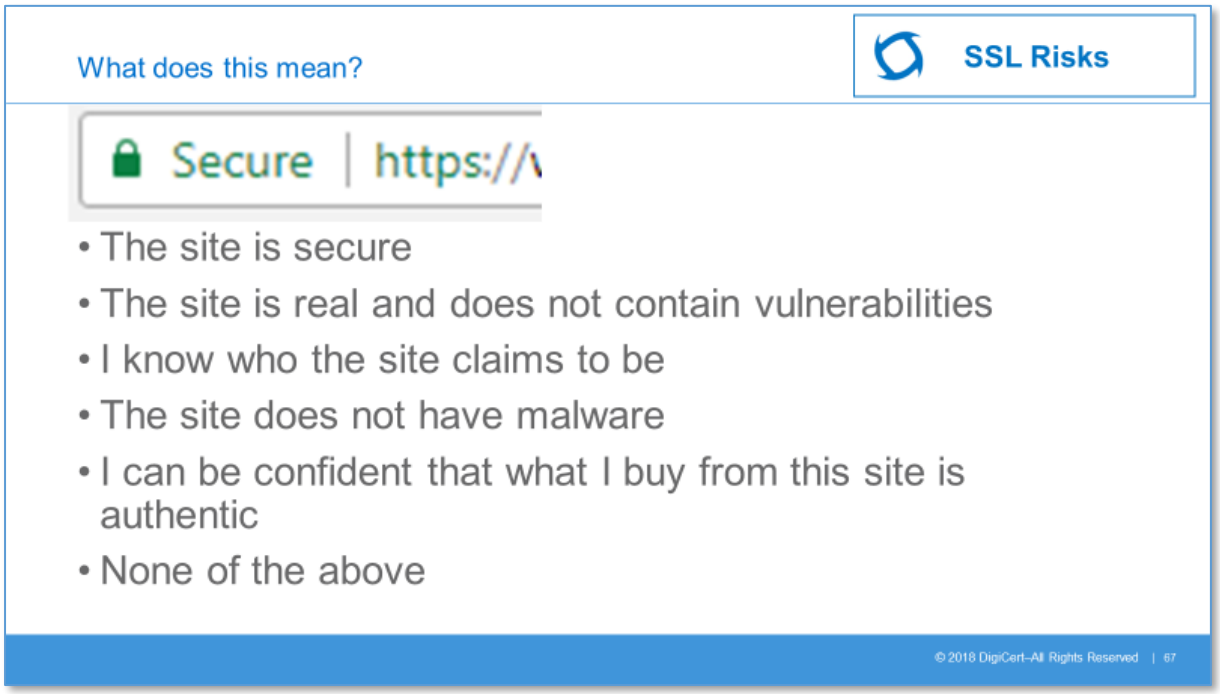
ROBOT is the return of a 19-year-old vulnerability that allows performing RSA decryption and signing operations with the private key of a TLS server. In 1998, Daniel Bleichenbacher discovered that the error messages given by SSL servers for errors in the PKCS #1 v1.5 padding allowed an adaptive-chosen ciphertext attack; this attack fully breaks the confidentiality of TLS when used with RSA encryption.

ROBOT only affects TLS cipher modes that use RSA encryption. Most modern TLS connections use an Elliptic Curve Diffie Hellman key exchange and need RSA only for signatures.



23 Feb 2017: Google researchers and academics demonstrated it is possible – following years of number crunching – to produce two different documents that have the same SHA-1 hash signature.

Phishing



The "Secure" padlock sign may not mean what you think. All it means is that the data is encrypted between your computer and the server, and that you are connected to that domain name.

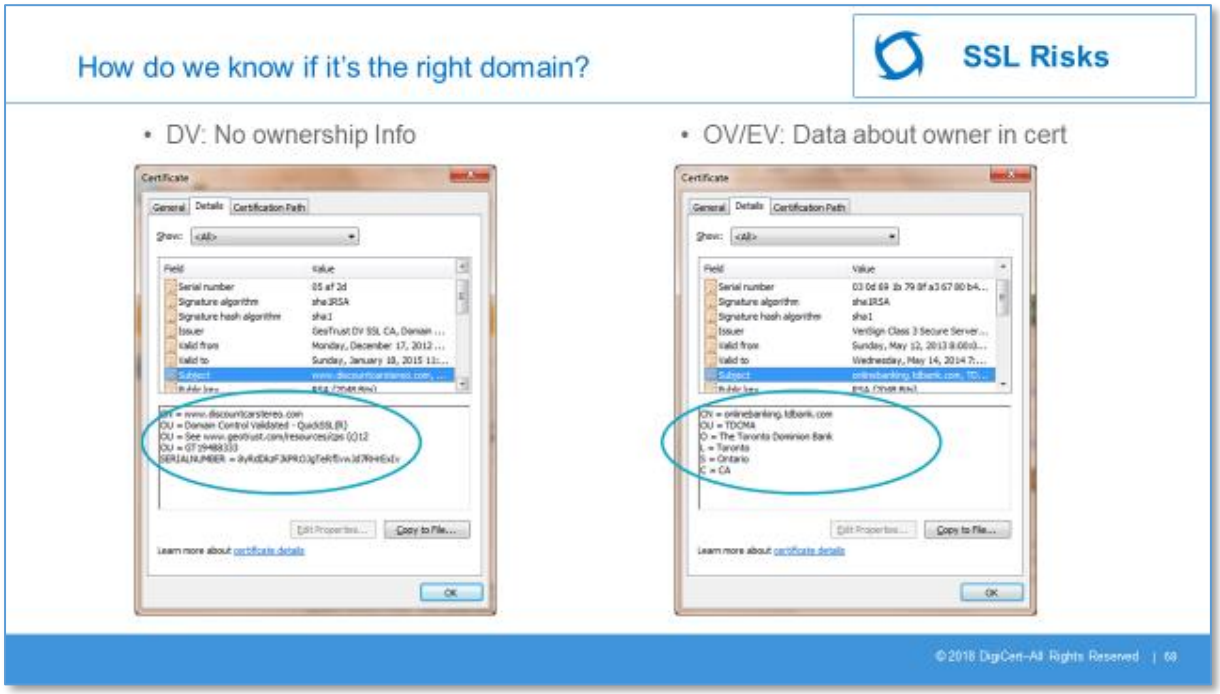
Phishing is still prevalent:

- Deceptions remain a common phisher convention.
- Visual deception - strings that look like a brand or familiar name - remain popular.

- Phishers obfuscate URLs; in particular, phishers mimic content delivery systems to generate "messy URLs" to deceive you (and to defeat antispam measures).

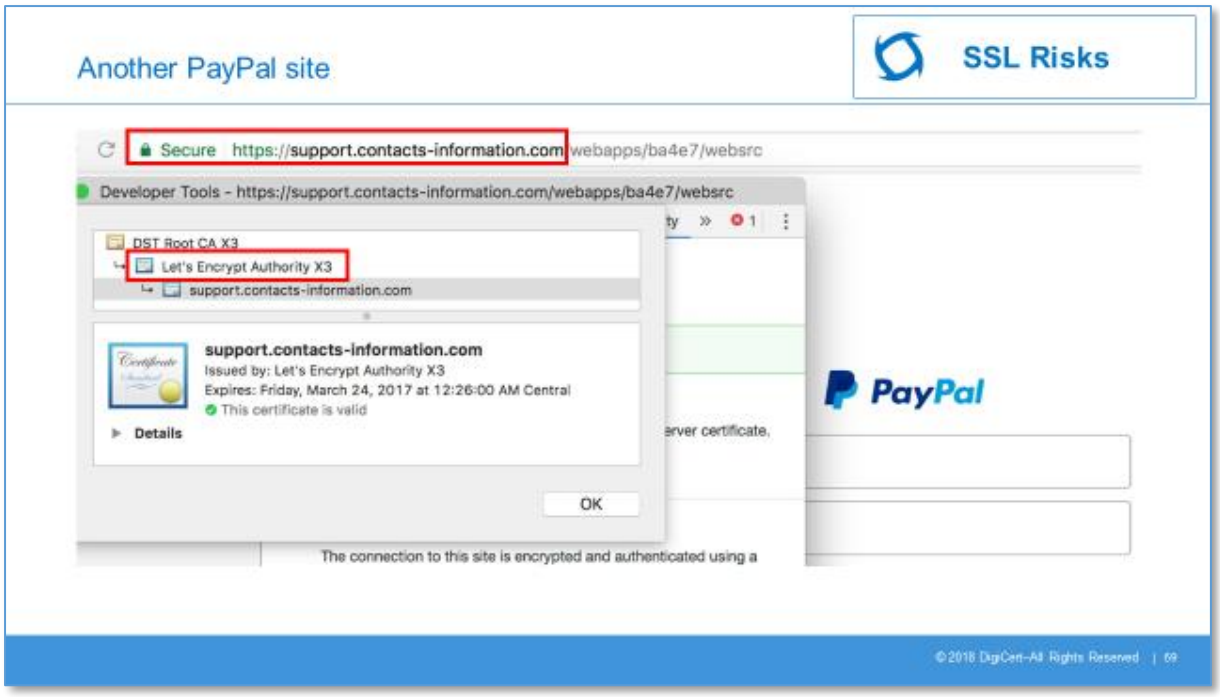
And the lessons to learn?

- Read the entire URL.
- Read what is displayed, not what you expect to be displayed.
- Take the time to distinguish the domain name from the URL.
- Resist the temptation to stop reading once you've encountered a brand or familiar string of characters in URLs and only use this to conclude the URL is safe.

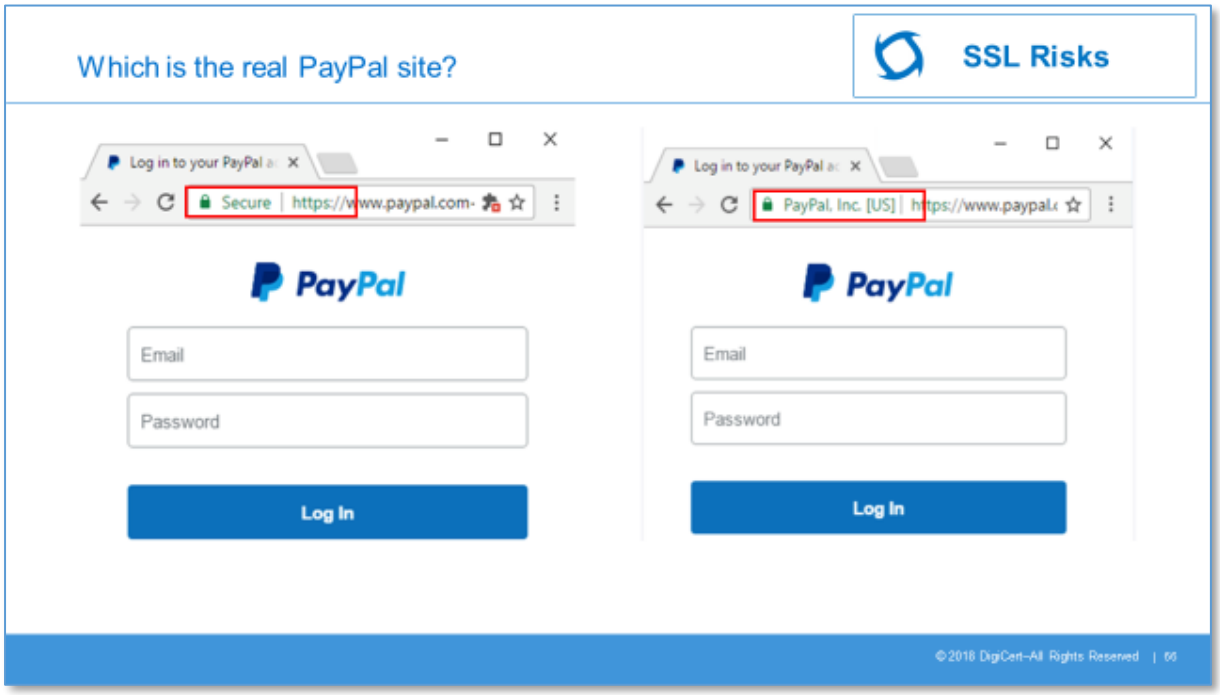


A Domain-validated (DV) certificate is a certificate that only includes your domain name in the certificate (not your business or organization name).

An Organization Validation (OV) displays two things that must be verified before you can be issued a high assurance certificate: ownership of the domain name and valid business registration. Both of these items are listed on the certificate, so visitors can be sure that you are who you say you are.



This phishing site looks like Paypal – but on closer inspection this is unlikely.



Certificate Authorities like DigiCert rely on the contact information maintained by domain registrars to determine domain ownership. They check on information about the owner of the domain and confirm that the applicant for a certificate is authorized to apply for and obtain the certificate.

A negligent Certificate Authority (CA) could issue a certificate to a fraudulent site. A spoofed domain name could go unnoticed, so there may even be a valid certificate issued to the MITM proxy. To address this concern, validation personnel are trained to look for similar, misleading domain names or ones that would otherwise be considered “high risk.” Also, because security is added to the server

certificate when the CA conducts more extensive checks on the organization applying for the certificate, the use of domain-only validated (DV) certificates is therefore discouraged.

An EV certificate is designed to prevent phishing attacks. It provides even greater assurance to customers than OV certificates by making the address bar turn green and displaying the organization name in the address bar.

Attacks on Certificate Authorities

There have been instances of a breach in security of networks of some Certifying Authorities. Comodo, DigiNotar, DigiSign & StartCom are some of those CAs. Hacker(s) have been reported exploiting common vulnerabilities within poorly maintained servers and firewalls. The hacker(s) have also been reported to have used advanced attack methods to penetrate the HSM (Hardware Security Module) with only one single open port.

Certificate Authority Attacks DigiNotar

SSL Risks

- DigiNotar announced on the 30th August 2011 that it had been breached which resulted in the fraudulent issuance of public key certificates "for a number of domains"
- On September 3rd, the Dutch government announced that because of the breach, "it could not guarantee the security of its own Web sites"
- In addition, the government said it was taking over DigiNotar's operations
- DigiNotar filed for bankruptcy on September 20th, 2011

9 Sep 2011 | 20:45 GMT

DigiNotar Certificate Authority Breach Crashes e-Government in the Netherlands

© 2018 DigiCert–All Rights Reserved | 72

Case Studies

Case Study: European Telco (2011)

SSL Risks

- The certificate that controlled the connection between all the company's retail shops and their ordering system unexpectedly expired
- No shop could order anything over the busy weekend period
- Estimated lost revenue as €200,000/hour for 5 hours

© 2018 DigiCert–All Rights Reserved | 74

This case study is based on a leading European Telco providing wireless and wired voice, internet and TV services to over 45M customers. The story starts late in 2011 - on a Friday night a certificate suddenly expired without anyone knowing it was there or who controlled it. This was the main certificate that controlled the connection between all the company's retail shops and their ordering system.

As this happened in their busy weekend period, this meant that they were losing approximately €200,000 an hour, as no shop could order anything while the connection was down. After about 5

hours, the problem was identified and resolved, however this was a major revenue hit and embarrassment for the company.

Then in 2012, the company had 2 major and well-publicised breaches by hackers. These events prompted a major review of all network security, including SSL.

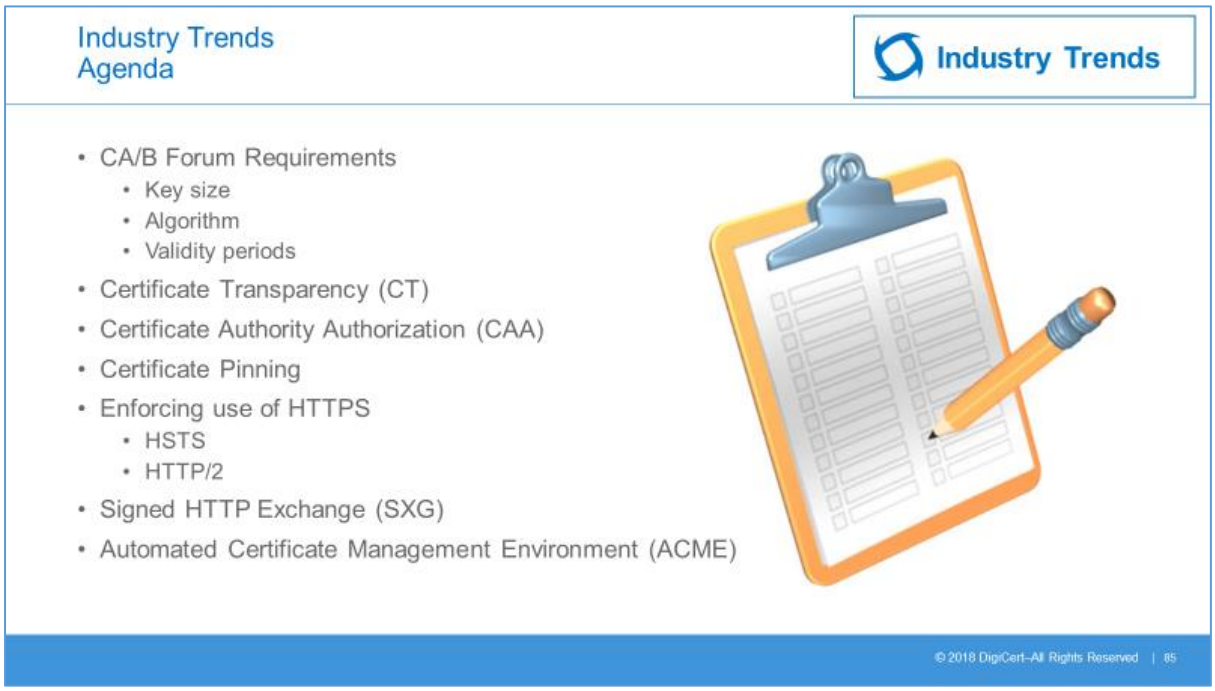
In December 2018, more than 30 million customers of mobile providers including O2, Tesco Mobile and Sky Mobile were unable to get online, with many also unable to make or receive phone calls, after a technical fault caused a UK-wide outage.

The problem hit O2's entire network and also affected companies that use its platform, including its subsidiary Giffgaff and Lycamobile.

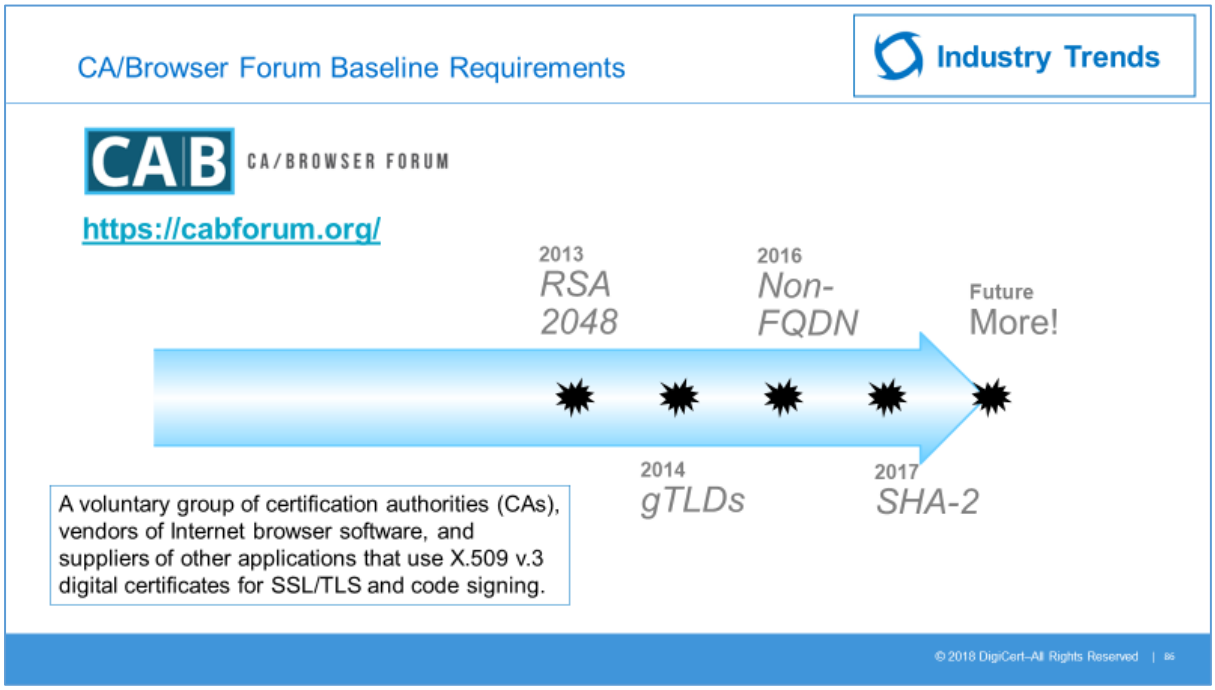
Swedish firm Ericsson admitted that its software had caused the problem. Ericsson president Börje Ekholm gave more detail about the cause of the disruption.

He said: "an initial root cause analysis" had indicated that the "main issue was an expired certificate in the software versions installed with these customers".

Industry Trends



CA/Browser Forum Requirements



Organized in 2005, The CA/Browser Forum is a voluntary group of certification authorities (CAs), vendors of Internet browser software, and suppliers of other applications that use X.509 v.3 digital certificates for SSL/TLS and code signing.

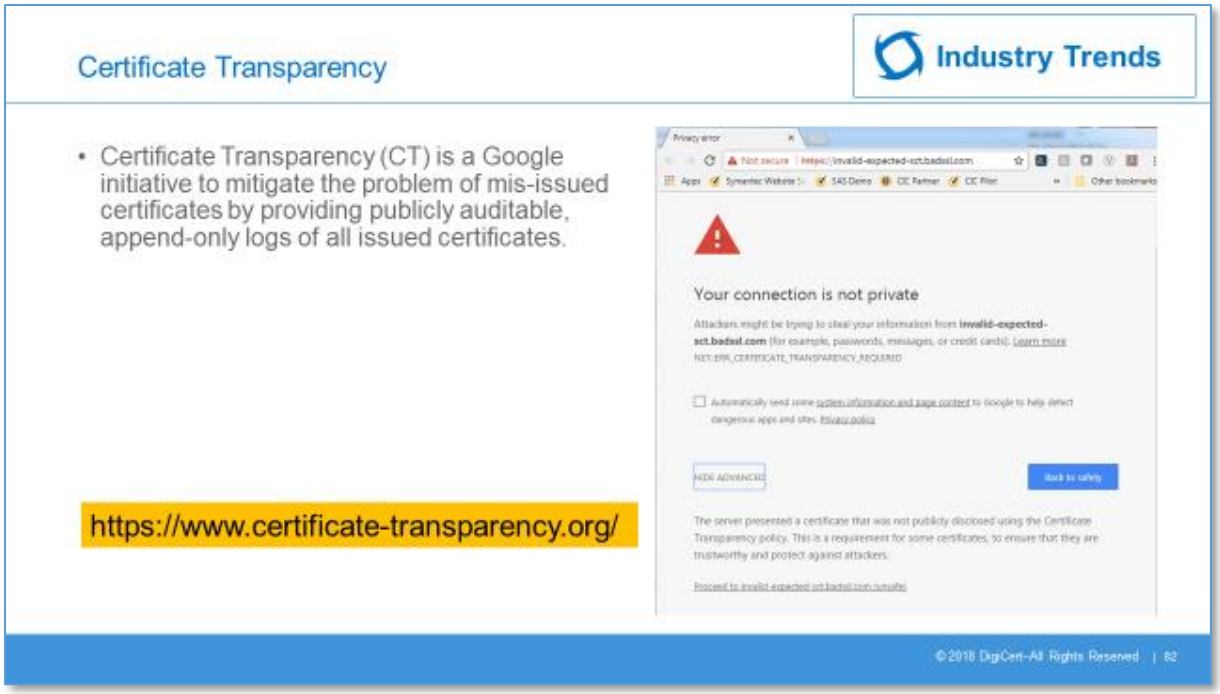
The CA/Browser Forum Baseline Requirements for the Issuance and Management of Publicly-Trusted Certificates describe a subset of the requirements that a certification authority must meet in

order to issue digital certificates for SSL/TLS servers to be publicly trusted by browsers. These requirements are continuously evolving to meet the ever-changing threat landscape. Recent changes include:

- **RSA 2048**: All certificates after 2013 must have public key size ≥ 2048 bits.
- **gTLDs**: CAs MUST provide a warning to the applicant that the gTLD may soon become resolvable and that, at that time, the CA will revoke the Certificate unless the applicant promptly registers the domain name. Within 30 days after ICANN has approved a new gTLD for operation, CA MUST cease issuing Certificates containing a Domain Name that includes the new gTLD until after the CA has first verified the Subscriber's control over or exclusive right to use the Domain Name
- **Non-FQDN**: After November 1, 2015 Certificates for Internal Names or IP addresses will no longer be trusted. As defined in the Baseline Requirements, the publicly-trusted SSL CAs will stop issuing certificates with non-FQDNs by November 1, 2015, and all unexpired certificates with non-FQDNs will be revoked by October 1, 2016. The CAs must also provide a warning of the deprecated use of such certificate to the applicant before issuance.
- **SHA-2**: All certificates after 2016 must use SHA-2 (SHA-256, SHA-384 or SHA-512) Digest Algorithm (hashing). (Note: a SHA-1 Root still permissible/recommended.)
- **Validity**: Maximum SSL validity period restricted to 2 years (825 days / 27 months) effective March 1, 2018. In February 2020, Apple announced that Safari will only trust certificates with a validity of 398 days or less (one year plus a renewal grace period). This policy goes into effect September 1, 2020. (Certificates issued before that date are not affected and do not need to be replaced or modified—you can continue to issue 2-year certificates until August 31, 2020, and use them until their expiration.)

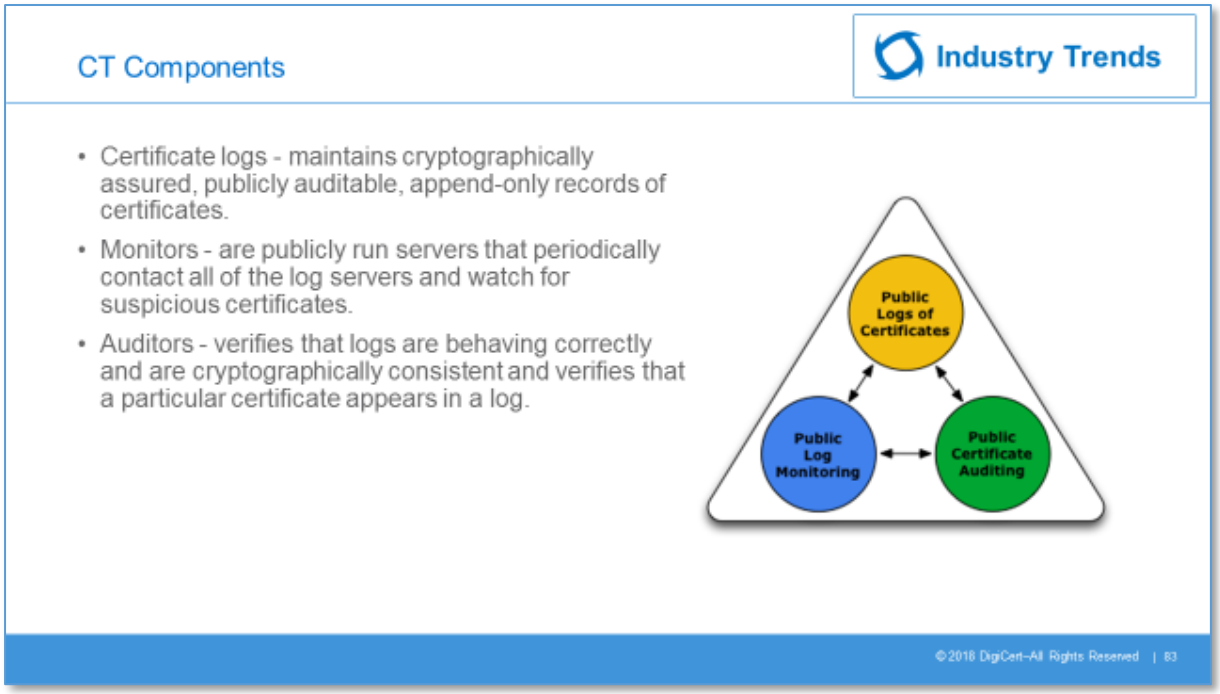
There will be more migrations and changes in the future as the CA/B Forum seeks to make the internet a safer place.

Certificate Transparency (CT)



Certificate Transparency (CT) first came on the radar a few years ago when Google announced it as a requirement for all Extended Validation (EV) SSL/TLS Certificates issued after 1 Jan 2015. Since then, Google has expanded the requirement to cover all types of SSL Certificates and most recently announced a deadline of April 2018. Certificates issued after that date that are not CT qualified will not be trusted in Chrome.

Apple enforced CT in late 2018. Certificates that fail to comply with this policy will result in a failed TLS connection, which can break an app's connection to Internet services or Safari's ability to seamlessly connect.

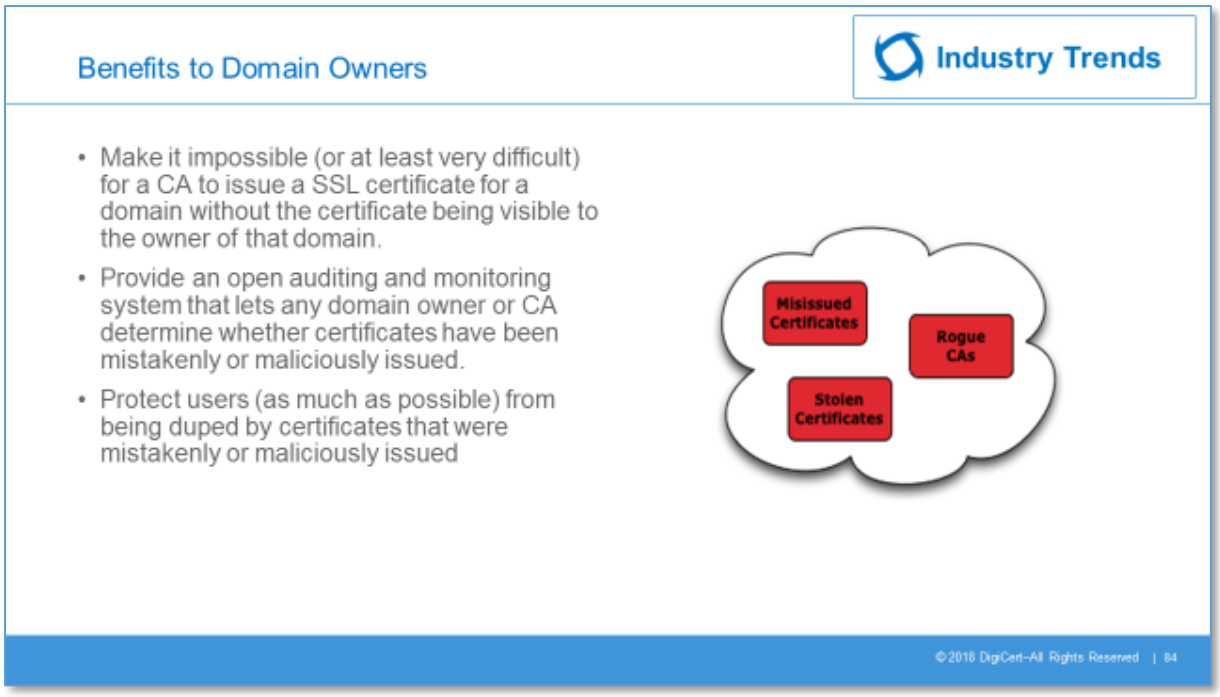


Certificate logs are simple network services that maintain cryptographically assured, publicly auditable, append-only records of certificates.

Monitors are publicly run servers that periodically contact all of the log servers and watch for suspicious certificates. For example, monitors can tell if an illegitimate or unauthorized certificate has been issued for a domain, and they can watch for certificates that have unusual certificate extensions or strange permissions, such as certificates that have CA capabilities.

A monitor acts much the same way as a credit-reporting alert, which tells you whenever someone applies for a loan or credit card in your name.

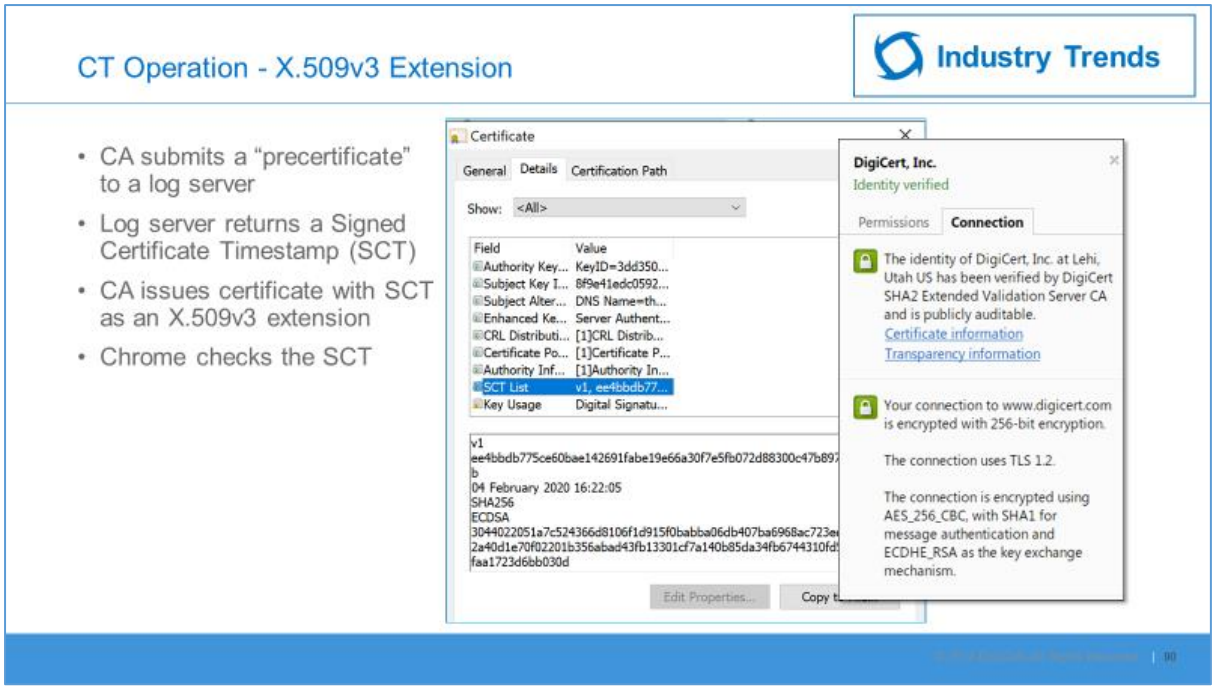
Auditors are lightweight software components that typically perform two functions. First, they can verify that logs are behaving correctly and are cryptographically consistent. If a log is not behaving properly, then the log will need to explain itself or risk being shut down. Second, they can verify that a particular certificate appears in a log.



Anyone can submit a certificate to a log, although most certificates will be submitted by certificate authorities and server operators. When someone submits a valid certificate to a log, the log responds with a signed certificate timestamp (SCT), which is simply a promise to add the certificate to the log within some time period. The time period is known as the maximum merge delay (MMD).

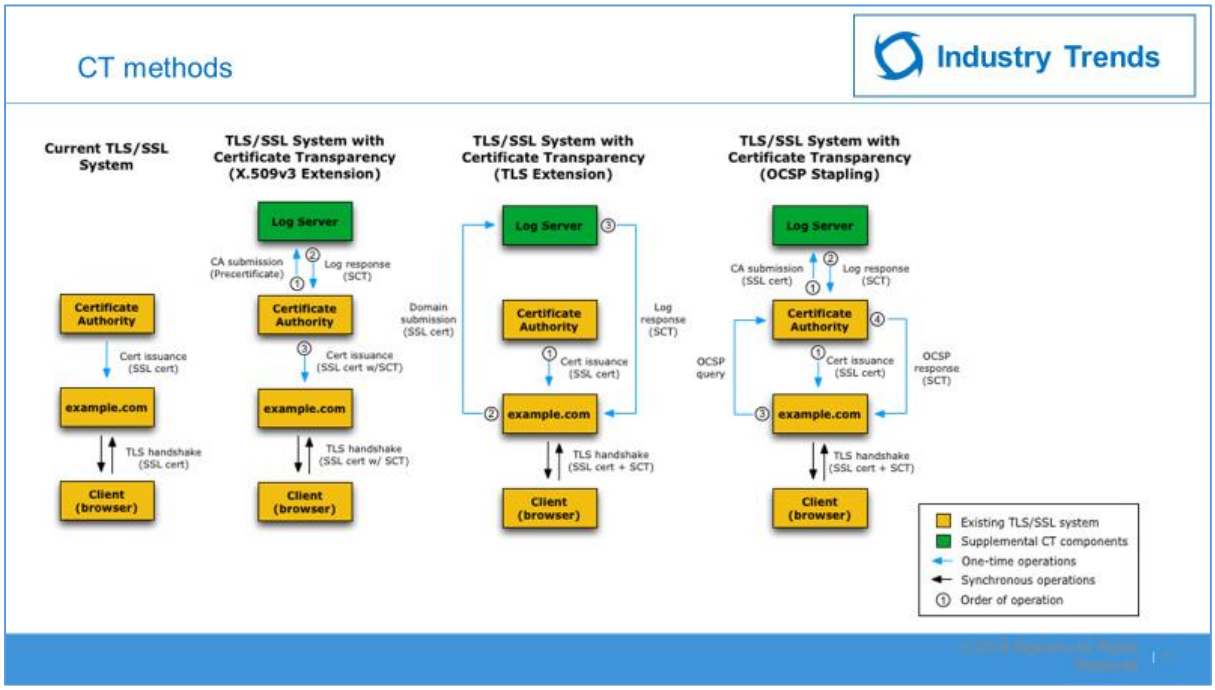
The MMD helps ensure that the log server adds the certificate to the log within a reasonable timeframe and doesn't block the issuance or use of the certificate, whilst allowing the log to run a distributed farm of servers for resilience and availability. The SCT accompanies the certificate throughout the certificate's lifetime. In particular, a TLS server must deliver the SCT with the certificate during the TLS handshake.

Certificate Transparency supports three methods for delivering an SCT with a certificate.



Certificate authorities can attach an SCT to a certificate using an X.509v3 extension. The certificate authority (CA) submits a precertificate to the log, and the log returns an SCT. The CA then attaches the SCT to the precertificate as an X.509v3 extension, signs the certificate, and delivers the certificate to the server operator.

This method does not require any server modification, and it lets server operators continue to manage their SSL certificates the same way they always have.



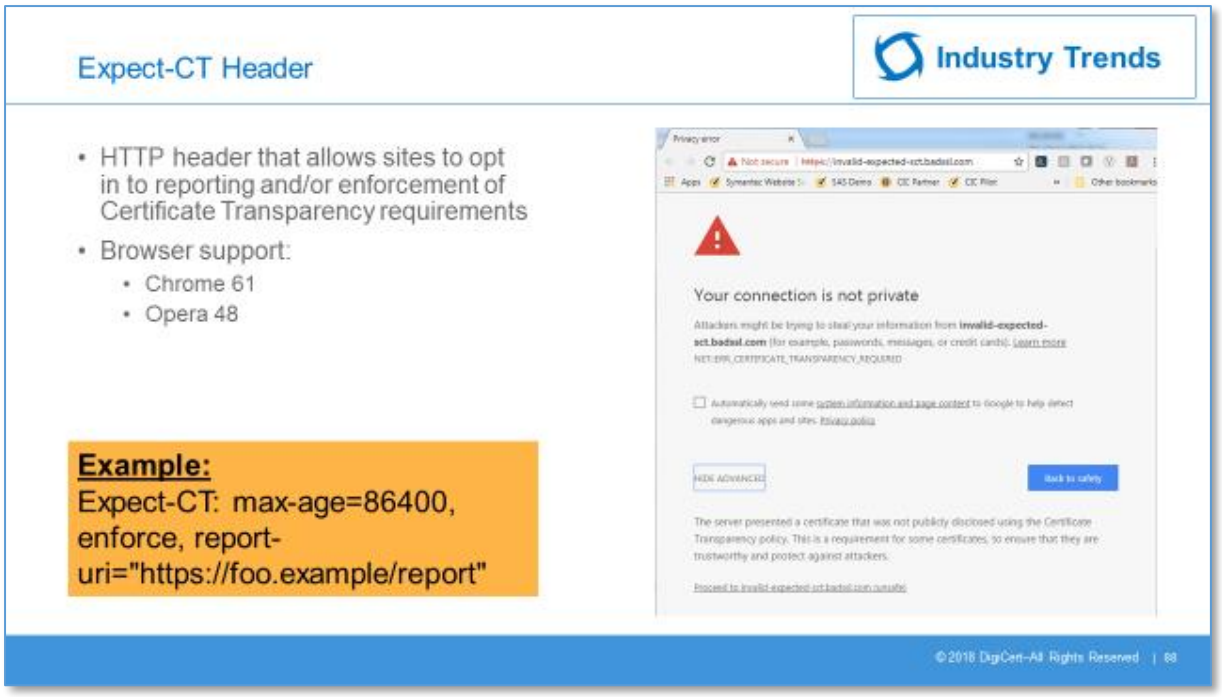
TLS Extension: Server operators can deliver SCTs by using a special TLS extension. In this case, the CA issues the certificate to the server operator, and the server operator submits the certificate to the log. The log sends the SCT to the server operator, and the server operator uses a TLS extension with type signed_certificate_timestamp to deliver the SCT to the client during the TLS handshake.

This method does not change the way a CA issues SSL certificates. However, it does require a server change to accommodate the TLS extension.

OCSP Stapling: Server operators can also deliver SCTs by using Online Certificate Status Protocol (OCSP) stapling. In this case, the CA simultaneously issues the certificate to the log server and the server operator. The server operator then makes an OCSP query to the CA, and the CA responds with the SCT, which the server can include in an OCSP extension during the TLS handshake.

This method allows CAs to take responsibility for the SCT but does not delay the issuance of the certificate, since the CA can get the SCT asynchronously. It does, however, require modification of the server to do OCSP stapling.

Above is an example result from a CT log search. Certificate Log servers are hosted by Cloudflare, DigiCert, Google and others. “Pilot” and “Skydiver” are both Google log servers. Utilities to scan log servers are freely available online.



This header allows web host operators to instruct user agents (browsers) to expect valid Signed Certificate Timestamps (SCTs) to be served on connections to these hosts. When configured in enforcement mode, user agents (UAs) will remember that hosts expect SCTs and will refuse connections that do not conform to the UAs Certificate Transparency policy.

The Expect-CT header allows sites to opt in to reporting and/or enforcement of Certificate Transparency requirements, which prevents the use of mis-issued certificates for that site from going unnoticed. When a site enables the Expect-CT header, they are requesting that the browser check that any certificate for that site appears in public CT logs.

By deploying the header but not enforcing it you can get feedback from the browser to see if it was satisfied with the Signed Certificate Timestamps it received. If there are problems, you can make sure they're resolved before the deadline and once you're ready to commit you can enforce the header to tell the browser to always expect and enforce CT.

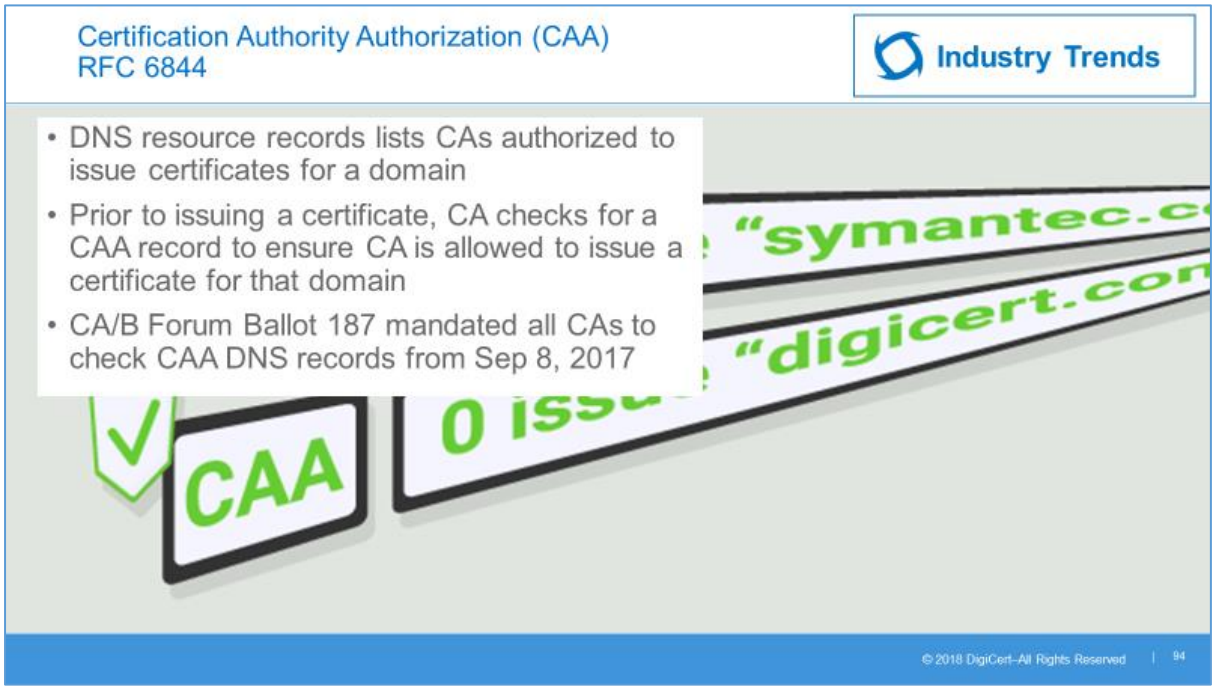
Directives:

- max-age Specifies the number of seconds after reception of the Expect-CT header field during which the user agent should regard the host from whom the message was received as a known Expect-CT host. If a cache receives a value greater than it can represent, or if any of its subsequent calculations overflows, the cache will consider the value to be either 2147483648 (2^31) or the greatest positive integer it can conveniently represent.
- report-uri="<uri>" (Optional) Specifies the URI to which the user agent should report Expect-CT failures.
- enforce (Optional) Signals to the user agent that compliance with the Certificate Transparency policy should be enforced (rather than only reporting compliance) and that the user agent should refuse future connections that violate its Certificate Transparency policy.

When both the enforce directive and the report-uri directive are present, the configuration is referred to as an "enforce-and-report" configuration, signalling to the user agent both that

compliance to the Certificate Transparency policy should be enforced and that violations should be reported.

Certificate Authority Authorization (CAA)

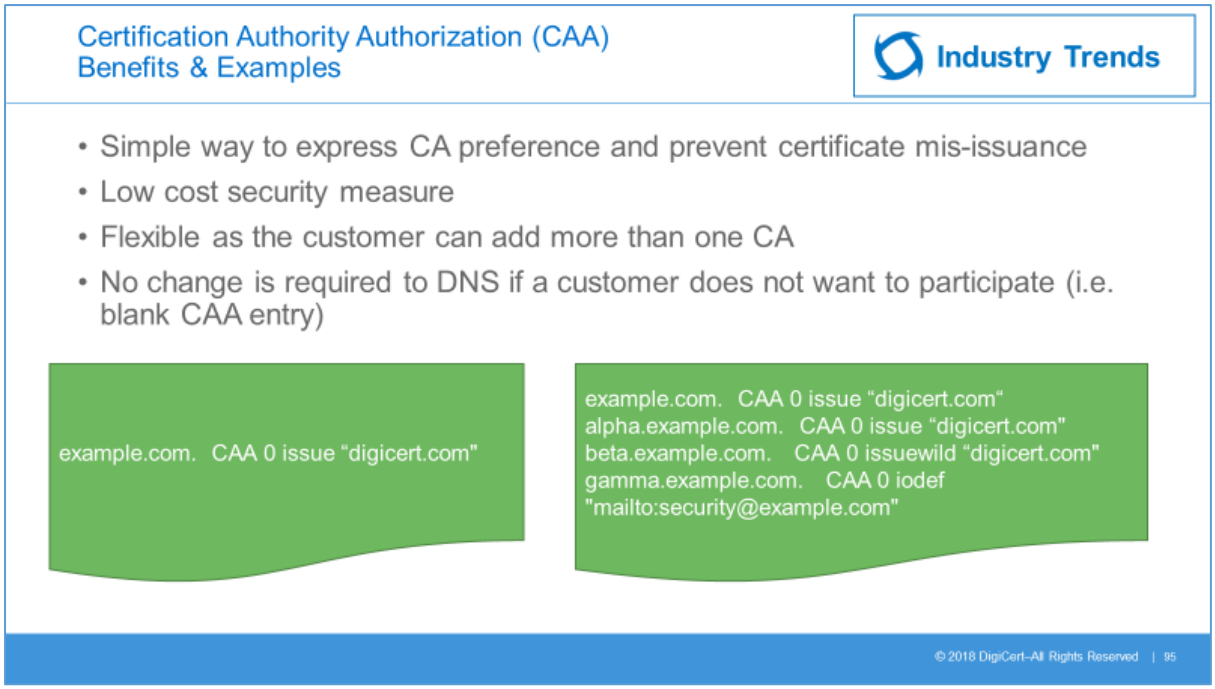


A DNS Certification Authority Authorization (CAA) record is used to specify which certificate authorities (CAs) are allowed to issue certificates for a domain.

The purpose of the CAA record is to allow domain owners to declare which certificate authorities are allowed to issue a certificate for a domain. They also provide a means for indicating notification rules in case someone requests a certificate from a non-authorized certificate authority. If no CAA record is present, any CA is allowed to issue a certificate for the domain. If a CAA record is present, only the CAs listed in the record(s) are allowed to issue certificates for that hostname.

CAA records can set policy for the entire domain, or for specific hostnames. CAA records are also inherited by subdomains, therefore a CAA record set on example.com will also apply to any subdomain, such as subdomain.example.com (unless overridden). CAA records can control the issuance single-name certificates, wildcard certificates, or both.

CA/B Forum Ballot 187 went into effect in September 2017 and states that all CAs must check CAA before issuing publicly trusted TLS certificates.



The DNS CAA record is specified by RFC 6844.

The canonical representation is: CAA <flags> <tag> <value>

Flag: An unsigned integer between 0-255. It is currently used to represent the critical flag, which has a specific meaning per RFC.

The RFC currently defines 3 available tags:

- issue: explicity authorizes a single certificate authority to issue a certificate (any type) for the hostname.
- issuewild: explicity authorizes a single certificate authority to issue a wildcard certificate (and only wildcard) for the hostname.
- iodef: specifies an URL to which a certificate authority may report policy violations.

Examples

To indicate that only the certificate authority identified by ca.example.net is authorized to issue certificates for example.com and all subdomains, one may use this CAA record:

example.com. IN CAA 0 issue "ca.example.net"

To disallow any certificate issuance, one may allow issuance only to an empty issuer list:

example.com. IN CAA 0 issue ";"

To indicate that certificate authorities should report invalid certificate requests to an email address and a Real-time Inter-network Defense endpoint:

example.com. IN CAA 0 iodef "mailto:security@example.com"

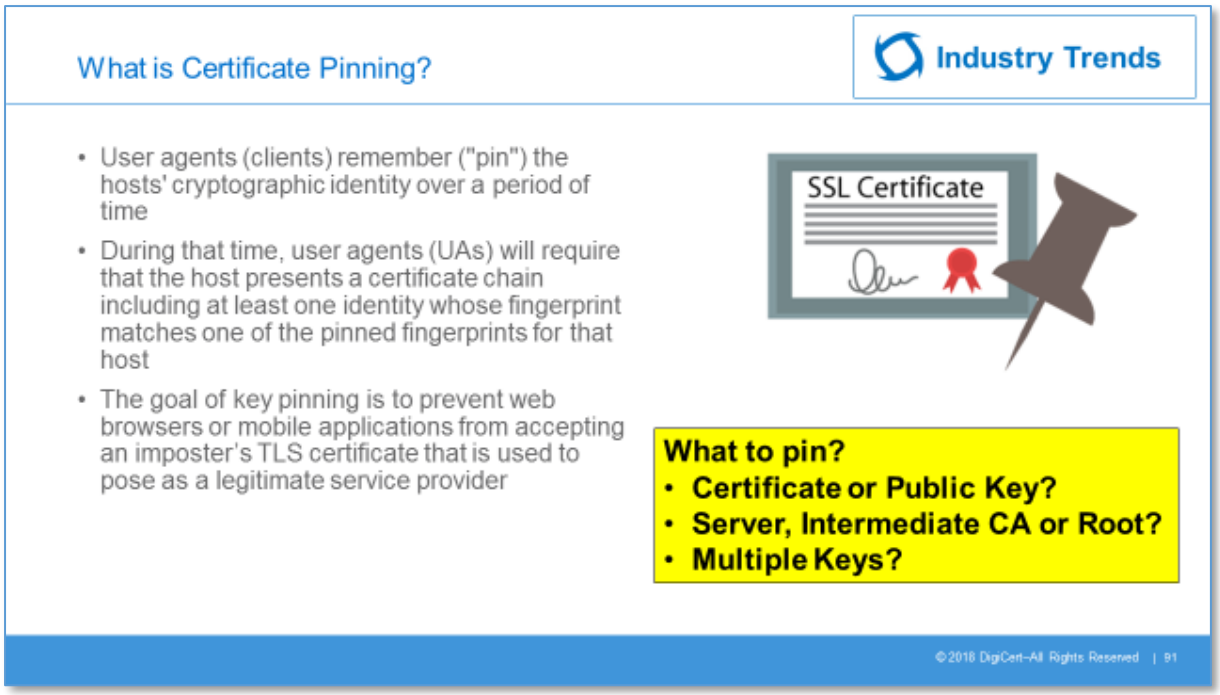
example.com. IN CAA 0 iodef http://iodef.example.com/

An Issuer MAY choose to specify parameters that further constrain the issue of certificates by that Issuer -- for example, specifying that certificates are to be subject to specific validation policies, billed to certain accounts, or issued under specific trust anchors.

For example, if ca1.example.net has requested that its customer account.example.com specify their account number "230123" in each of the customer's CAA records using the (CA-defined) "account" parameter, it would look like this:

account.example.com CAA 0 issue "ca1.example.net; account=230123"

Certificate Pinning



Certificate Pinning is the process of associating a host with their expected X509 certificate or public key. Once a certificate or public key is known or seen for a host, the certificate or public key is associated or 'pinned' to the host. If more than one certificate or public key is acceptable, then the program holds a pinset. In this case, the advertised identity must match one of the elements in the pinset.

A host or service's certificate or public key can be added to an application at development time, or it can be added upon first encountering the certificate or public key. The former - adding at development time - is preferred since preloading the certificate or public key out of band usually means the attacker cannot taint the pin.

Certificate pinning can substantially improve the security of the SSL/TLS and PKI ecosystem. There are 3 key security benefits:

- Attack surface reduction: The dominant TLS authentication model in use today relies on public CAs. Their job is to issue certificates to domain name owners but not to other random people. In turn, user agents trust all CA-issued certificates unconditionally. This model suffers from an enormous flaw: any CA can issue a certificate for any domain name. Given that there are hundreds of CAs and possibly thousands of entities who influence certificate issuance in one way or another, the attack surface is huge. With pinning, owners can specify (pin) the CAs that are allowed to issue certificates for their domain names.

- Key continuity: Key continuity is a variation on the previous use case, but it can be used without relying on public CAs. Let's assume that you somehow know that a particular key is valid for some web site. With that, whenever you visit the site you can compare their current key with your "correct" key; if the keys match, you know that you are not under attack. (Key continuity is commonly used with the SSH protocol.) Keys are associated with servers when they are seen for the first time and checked on subsequent visits.
- Authentication: Pinning can even be used for authentication, provided there is a reliable (secure) channel to communicate the required cryptographic identities to end users.

Pinning is therefore a good idea in theory, however the next question is: what to pin? There are a number of options, all with advantages and disadvantages. In general, pin to cryptographic identity for greatest security (e.g. pin to the public key or a hash thereof); pin to certificate metadata for greatest flexibility (e.g. subjectDN.OU=$X). Pin to both as part of fall-back logic if it makes sense in context.

In many cases, the public key hash is used as the pin data. But there is still the question of which certificate public key should be used!

- Pin the server ("leaf") certificate: This is most secure, however the certificate will expire in 2 years (or sooner) so you need a way to seamlessly transition to a new certificate.
- Pin the intermediate certificate: This is less secure than pinning the leaf, and the intermediate that you pin can still issue rogue certificates.
- Pin the root certificate: The same issues as pinning the intermediate, however the root will have a longer lifetime.

Public key pinning started at Google, and they first implemented in Chrome, pinning their own web sites. Their approach is an example of static pinning; the pins are not easy to change because they're embedded in the browser. Chrome's pinning has served us well over the years, uncovering many cases of fraudulent certificates that would otherwise perhaps fly under the radar. Google also

allowed (and still does) that other organisations embed their pins in Chrome. These days, Firefox also supports static pinning, drawing from the same pins maintained by Google.

Dynamic Pinning
HTTP Public Key Pinning (HPKP)

Industry Trends

- RFC 7469 (2015) requires a SHA-256 hash of one or more public keys
- To enable this feature for your site, you need to return the Public-Key-Pins HTTP header when your site is accessed over HTTPS:

```
Public-Key-Pins: pin-sha256="<pin-value>";
                 max-age=<expire-time>;
                 includeSubDomains;
                 report-uri="<uri>"
```

©2018 DigiCert-All Rights Reserved | 93

HTTP Public Key Pinning (HPKP) is a security feature that can prevent fraudulently issued TLS certificates from being used to impersonate existing secure websites.

HPKP Problems

Industry Trends

- Zero max-age
- Wrong pin directives
- Only one pinned public key
- No pins at all

Only 0.09% of the certificates in Netcraft's March 2016 SSL Survey are served with HPKP headers, which equates to fewer than 4,100 certificates in total

More surprisingly, around a third of these sites are using the HPKP header incorrectly, which effectively disables HPKP

HPKP removed from Chrome 68

Over Chrome

Google Chrome

Versie 68.0.3440.17 (Officiële build) beta (64-bits)

©2018 DigiCert-All Rights Reserved | 94

HPKP has a number of problems:

Zero max-age: Every HPKP policy must specify a max-age directive, which suggests how long a browser should regard the website as a "Known Pinned Host". The most commonly used max-age

value is 5184000 seconds (60 days). Nearly 1,200 servers use this value, while around 900 use 2592000 seconds (30 days).

Unfortunately, some sites feature pointlessly short max-age values, such as 5 or 10 seconds. These durations are far too short to be effective, as a victim's browser would rapidly forget about these known pinned hosts.

Additionally, a few sites explicitly specify a max-age of zero along with their public key pins. These sites are therefore not protected by HPKP and are in some cases needlessly sending this header to every client request. It is possible that they are desperately trying to remove a previously set HPKP policy, but this approach obviously cannot be relied upon to remove cached pins from browsers that do not visit the site in the meantime. These sites would therefore have to continue using a certificate chain that conforms to their previous HPKP policy or run the risk of locking out a few stragglers.

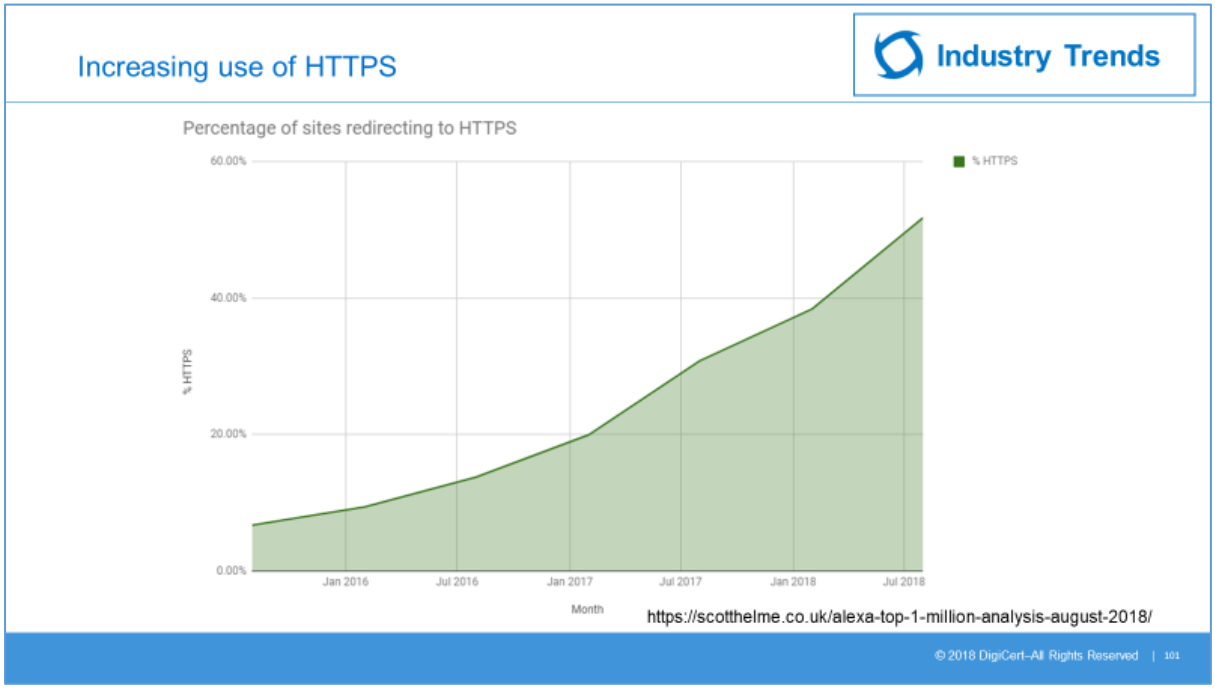
Wrong pin directives: Each pinned public key must be specified via a separate pin-sha256 directive, and each value must be a SHA256 hash; but more than 1% of servers that try to use HPKP fail to specify these pins correctly.

Only one pinned public key: It is essential that a secure site should specify at least two public key pins when deploying HPKP. At least one of these should be a backup pin, so that the website can recover from losing control of its deployed certificate. If the website owner still possesses the private key for one of the backup certificates, the site can revert to using one of the other pinned public keys without any browsers refusing to connect. However, many servers that use HPKP specify only one public key pin.

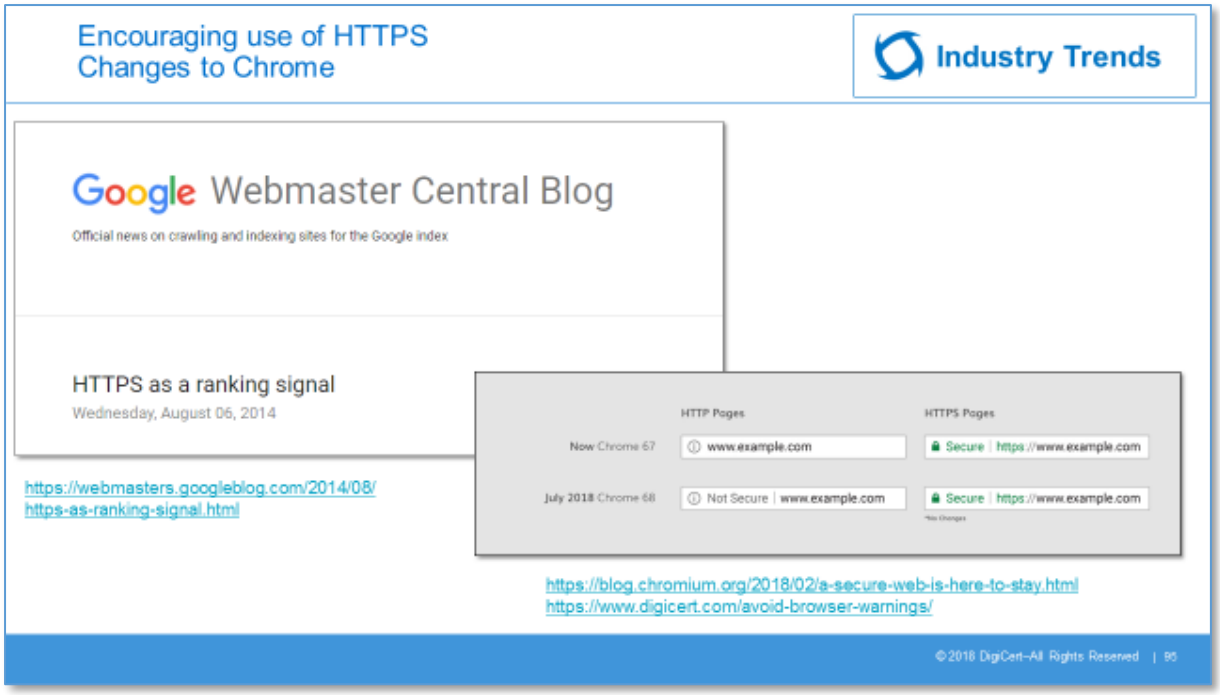
No pins at all: Some HPKP headers neglect to specify any pins at all, and a few try to set values that are not actually hashes (which has the same effect as not setting any pins at all).

Because of the problems associated with HPKP, Google removed support for HPKP from Chrome 68 onwards. It is reported that Google wants the ***Expect-CT*** header to replace HPKP.

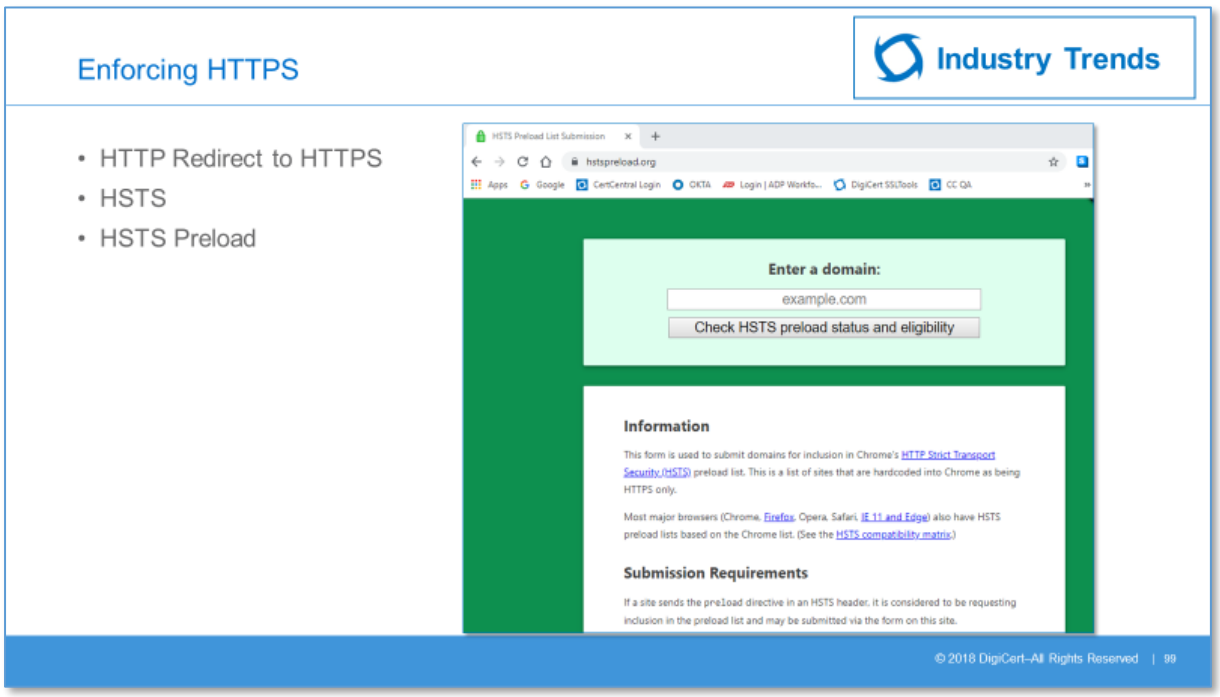
Enforcing HTTPS



September 2018: More than one-half (51.8 percent) of the one million most visited websites worldwide now actively redirect to HTTPS, the secure version of the HTTP protocol over which data between a device and a website is transmitted, according to stats by security researcher Scott Helme.



With the release of the Google Chrome 68 browser, any web page not running HTTPS with a valid TLS certificate will show a 'not secure' warning in the Chrome address bar. This warning will apply to Internet-facing websites and potentially millions of corporate/private intranet sites accessed through Chrome, which has about 60% market share, according to publicly available data.



A server-side redirect is a method of URL redirection using an HTTP status code (e.g. 301 Moved Permanently, 303 See Other, and 307 Temporary Redirect) issued by a web server in response to a request for a particular URL. The result is to redirect user's web browser to another web page with a different URL.

This approach is not secure as a nefarious “man in the middle” would instead substitute a phishing page. This hacker could capture your network traffic over HTTP for any website that relies on 301 redirects alone for switching from HTTP to HTTPS. This method presents a window of opportunity for the hacker to strip down your SSL encryption and steal valuable data or even worse, present a fake login portal page.

HTTP Strict Transport Security (HSTS) is a web server directive that informs user agents and web browsers how to handle its connection through a response header sent at the very beginning and back to the browser. This sets the Strict-Transport-Security policy field parameter. It forces those connections over HTTPS encryption, disregarding any script's call to load any resource in that domain over HTTP. HSTS forces browsers and app connections to use HTTPS if that is available. Even if someone just types in the www or http://.

HSTS preloading is a function built into the browser whereby a global list of hosts enforce the use of HTTPS ONLY on their site. This list is compiled by Chromium Project and is utilized by Chrome, Firefox and Safari. These sites do not depend on the issuing of the HSTS response headers to enforce the policy. Instead, the browser is already aware that the domain name requires the use of HTTPS ONLY and pushes HSTS before any connection or communication even takes place.

This removes the opportunity an attacker has to intercept and tamper with redirects over HTTP. The HSTS response header is still needed in this scenario and must be left in place for those browsers that don’t use preloaded HSTS lists.

The site shown above (https://hstspreload.org/) is run by Chromium and by submitting a domain here you’re asking for it to be preloaded into the browser itself. In other words, when the browser ships then your site will already be specified as only being accessible over HTTPS even if you’ve never visited it before. This means that the risk described above where the first request is insecure and HSTS is dependent on a secure response is gone – the browser will internally redirect to the secure scheme before anything hits the wire.

Enforcing HTTPS
HTTP Strict Transport Security (HSTS - RFC 6797)

Industry Trends

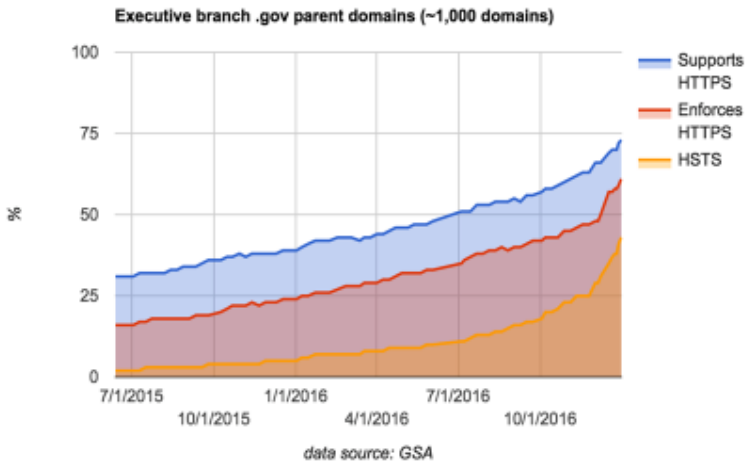
- Websites accessible only via secure connections
- "Strict-Transport-Security" entry is included in the HTTP response header
- Consider warning site visitors if they are using older versions of browsers
  - Browsers earlier than IE 8, Firefox 14, Safari 5, or Chrome 26 do not support HSTS

©2016 DigiCert-All Rights Reserved | 96

HSTS is a way to tell a web browser to always connect to a domain over https, so that even if a page somewhere says, "connect by http", the browser will switch it over https. It is like an insurance policy that ensures the use of HTTPS for your web site in case of an intentional or unintentional attempt to direct the user to an unencrypted http resource. It is described in RFC 6797 and is supported by all recent browser versions. To implement HSTS, "Strict-Transport-Security" entry is included in the HTTP response header. This is then cached by the browser to identify which sites are "HTTPS Only," and it is used by the browser when opening up subsequent sessions with the server's domain. This helps prevent subsequent man-in-the-middle attacks.

Increasing use of HTTPS
HTTPS across .gov

Industry Trends



©2016 DigiCert-All Rights Reserved | 99

US Presidential Memorandum M-15-13 (June 2015) calls for "all publicly accessible Federal websites and web services" to only provide service through a secure connection (HTTPS), and to use HTTP Strict Transport Security (HSTS) to ensure this. This applies to all public domains and subdomains operated by the federal government, regardless of the domain suffix, as long as they are reachable over HTTP/HTTPS on the public internet.

https://obamawhitehouse.archives.gov/sites/default/files/omb/memoranda/2015/m-15-13.pdf

Each public website or web service an agency operates must:

- Provide service over HTTPS.
- Automatically redirect HTTP requests to HTTPS, or disable HTTP entirely.
- Have an HSTS policy in place

https://https.cio.gov/guide/#compliance-and-best-practice-checklist

As of Spring 2017, all new .gov domains will be submitted to browsers for "preloading". Browsers will strictly enforce https.

"Always-on" SSL

"Always-on" SSL

Industry Trends

	Protection/Assurance		
Security Feature	**Good**	**Better**	**Best**
Persistent HTTPS	X	X	X
Cookies Set With *Secure* Flag	X	X	X
Extended Validation Certificates		X	X
HTTP Strict Transport Security (HSTS)			X

Source: OTA: Protecting Your Website With Always On SSL (May 2012)

- Avoid SSL vulnerabilities
  - Firesheep
  - SSL Strip
- Better search ranking
- Increased conversion rates
- Better security

© 2018 DigiCert–All Rights Reserved | 104

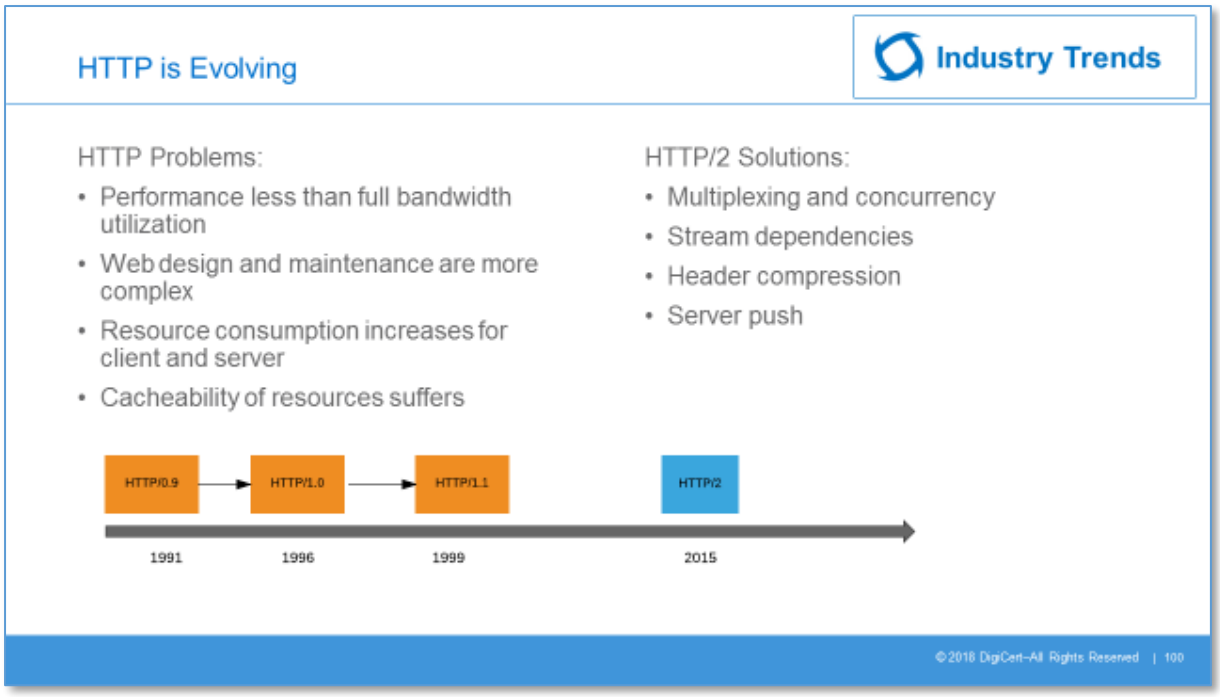
Companies who are serious about protecting their customers and their business reputation long term will implement "Always-On" SSL. The OTA has outlined the steps you can take to implement Always-On SSL and protect your users. The level of protection and assurance you can provide depends on the security features you choose to implement, as summarized in the table.

Enforcing "Persistent HTTPS" includes the following measures:

- Remove Mixed Content (replace HTTP references with HTTPS pointers)
- Install and test SSL certificates
- Fix Server Protocol and Cipher suite Settings
- Redirect HTTP traffic to HTTPS
- Implement an automated scanning system to detect and alert non-compliant sites

Another important measure is set the *Secure Flag* for all session cookies. A session cookie can be set with an optional "secure" flag, which tells the browser to contact the origin server using only HTTPS whenever it sends back this cookie. The Secure attribute should be considered security advice from the server to the user agent, indicating that it is in the session's interest to protect the cookie contents. This measure helps prevent cookies from being sent over HTTP, even if the user accidentally makes (or is tricked into making) a browser request to the Web server via HTTP.

HTTP/2



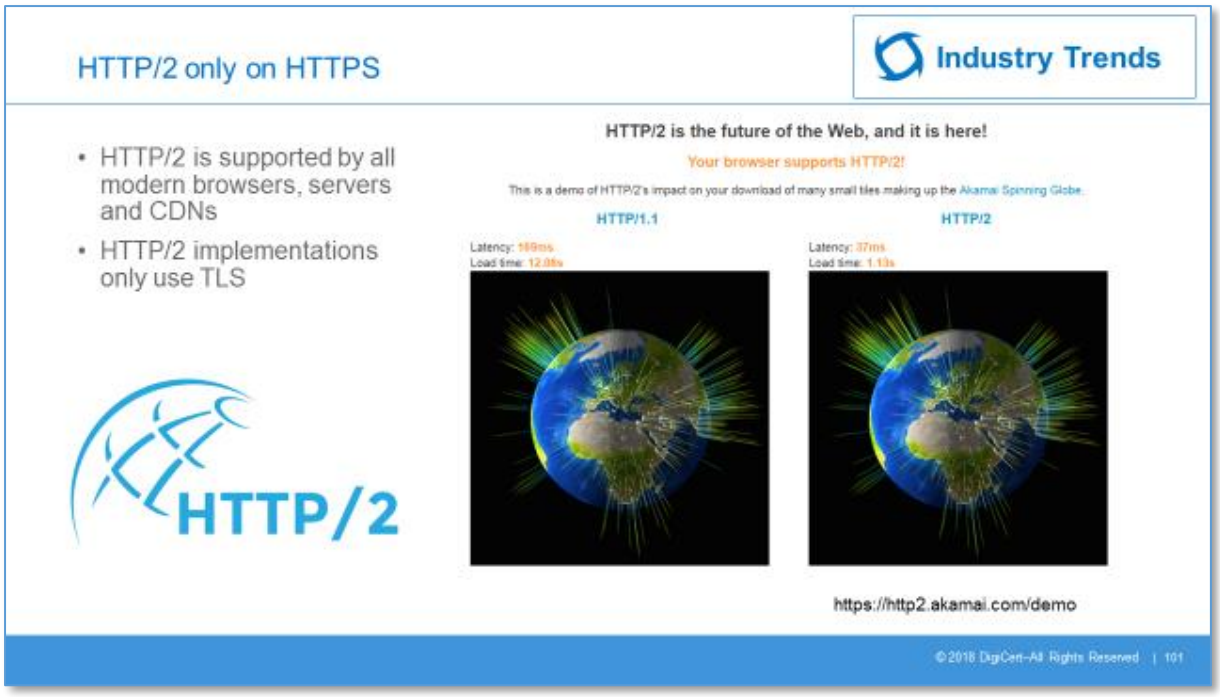
Tim Berners-Lee proposed initial HTTP protocol in 1991. The web has dramatically evolved over the last 20+ years, yet HTTP - the workhorse of the Web - has not. Web developers have worked around HTTP's limitations, but:

- Performance still falls short of full bandwidth utilization
- Web design and maintenance are more complex
- Resource consumption increases for client and server
- Cacheability of resources suffers

HTTP/2 attempts to solve many of the shortcomings and inflexibilities of HTTP/1.1. Its many benefits include:

- Multiplexing and concurrency: Several requests can be sent in rapid succession on the same TCP connection, and responses can be received out of order - eliminating the need for multiple connections between the client and the server
- Stream dependencies: the client can indicate to the server which of the resources are more important than the others
- Header compression: HTTP header size is drastically reduced
- Server push: The server can send resources the client has not yet requested

The HTTP/2 specification was published as RFC 7540 in May 2015.



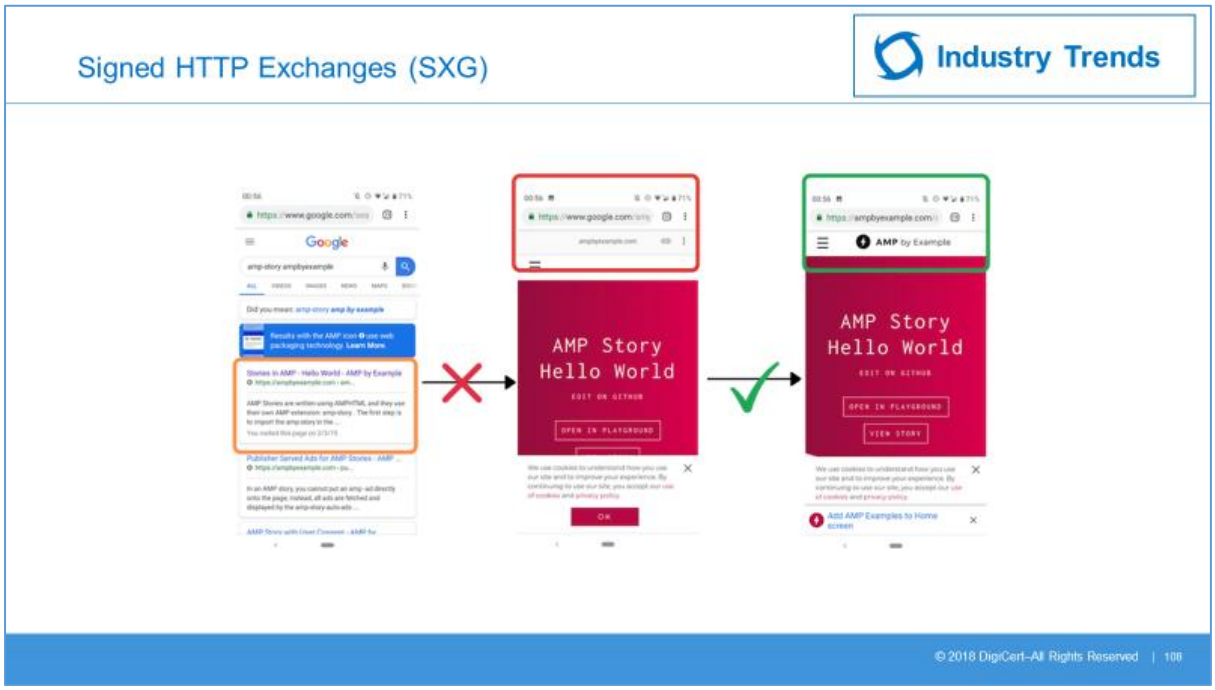
HTTP/2 is defined for both HTTP URIs (i.e. without encryption) and for HTTPS URIs (over TLS, where TLS 1.2 or newer is required).

Although the standard itself does not require usage of encryption, most client implementations (Firefox, Chrome, Safari, Opera, IE, Edge) have stated that they will only support HTTP/2 over TLS, which makes encryption de facto mandatory.

According to W3Techs, as of February 2020, 43.0% of the top 10 million websites supported HTTP/2.

HTTP/3 is the upcoming third major version of HTTP. HTTP/3 is a draft based on a previous RFC draft, then named "Hypertext Transfer Protocol (HTTP) over QUIC". QUIC is a transport layer network protocol developed initially by Google.

Signed HTTP Exchanges (SXG)



Google AMP (Accelerated Mobile Pages) is a website publishing technology that lets you create web pages that load almost instantly on mobile phones.

To build AMP pages, you need to create another version of your site that follows the AMP project's standards. Once you do that, your AMP site will have its own URL (site.com/page/amp) and be compatible with most popular web browsers like Chrome, FireFox, and Safari.

AMP provides speed benefits above and beyond the format through techniques like caching and preloading. These benefits can have downsides like extra URLs being displayed when embedded inside an AMP Viewer. By serving AMP content using *signed exchanges*, you can use a new web platform feature to overcome all of these.

Signed Exchange (or "SXG") is an emerging technology which provides a way to prove the authenticity of a web document. This can be used to determine a page's original publisher, no matter where the document is served from.

A publisher can "sign" an HTTP request-response pair with their domain's certificate. With it, the browser can safely show the publisher's URL in the address bar because the signature proves that the content originally came from the publisher's origin.

Simply put, Signed-Exchange = A Web Page + Certificate of its Origin

A signed exchange is made up of a valid AMP document and the original URL of the content. This information is protected by digital signatures that securely tie the document to its claimed URL. This enables browsers to safely display the original URL in the URL bar instead of the hostname of the machine that delivered the bytes to the browser. Signed AMP content is delivered in addition to (rather than instead of) regular AMP content.

Implementing SXG

For signing the HTTP request-responses for AMP pages, you need to get a Digital Certificate issued for your domain.

Implementing SXG

Industry Trends

1. Obtain an **ECDSA** TLS Certificate with **CanSignHttpExchanges** extension and **90-day** maximum validity
2. Update your DNS **CAA** record:

```
example.com. IN CAA 0 issue "digicert.com; cansignhttpexchanges=yes"
```

May 21, 2019 | Clint Wilson

Categories: SSL, CertCentral

Google's Signed HTTP Exchange Solution Displays Publisher URLs for AMP Pages via TLS

DigiCert first and only public CA to issue TLS certificates that enable Signed HTTP Exchange

© 2018 DigiCert–All Rights Reserved | 109

Note: The private key must be ECDSA, curve secp256r1. And the certificate should have the CanSignHttpExchanges extension enabled for production use. Per industry standards, certificates that include the Signed HTTP Exchange extension have a 90-day maximum validity limit.

For a Certificate Authority (CA) to issue your certificate with the CanSignHttpExchanges extension, you must do a one-time set up in the domain's DNS record and add the "cansignhttpexchanges=yes" parameter to the record:

```
example.com. IN CAA 0 issue "digicert.com; cansignhttpexchanges=yes"
```

Prior to issuing your certificate with the CanSignHttpExchanges extension, a CA (such as DigiCert) checks the domain's CAA resource record for a valid property with this parameter. If the record contains the "cansignhttpexchanges=yes", we can issue the certificate. If the domain doesn't have a CAA resource record or if the record doesn't contain this parameter, we can't issue the certificate.

Next, you need to run a "Packager" server, something which will sign (or, "package") the required pages using your certificate and its private key.

Delegated Credentials

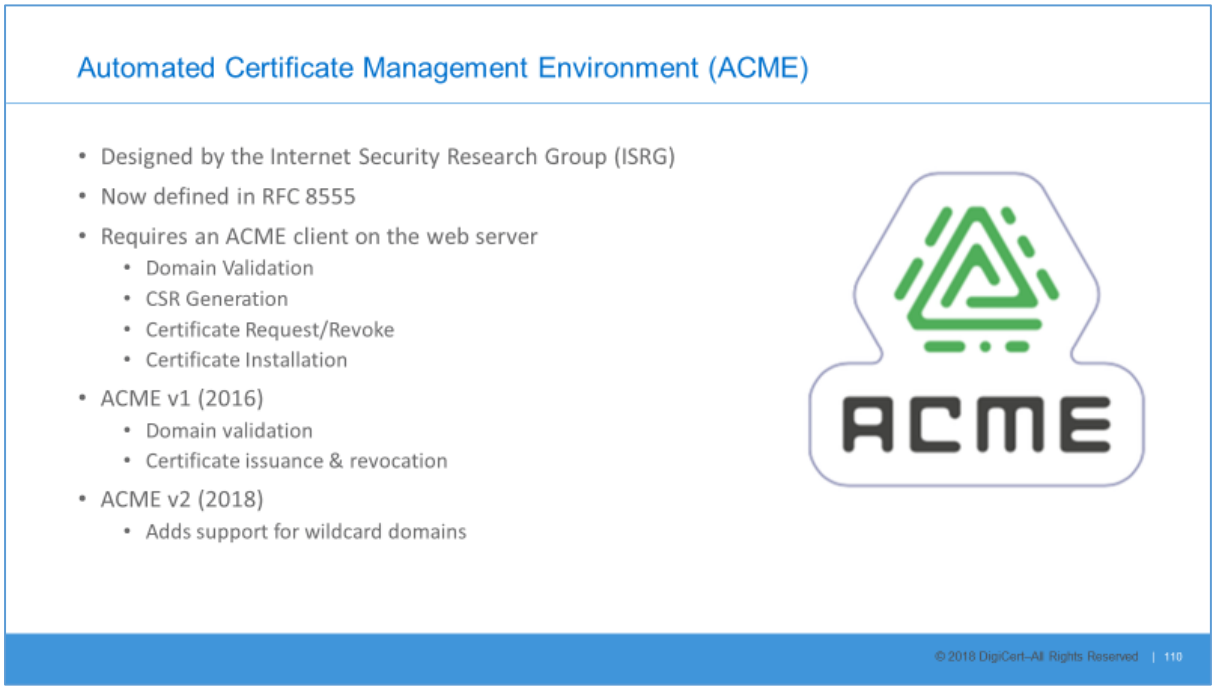
A delegated credential (DC) is a short-lasting key that the certificate's owner has delegated for use in TLS. DCs are used by CDNs (e.g. CloudFlare) to reduce latency and improve performance and reliability.

A delegated credential contains a public key and an expiry time. This bundle is then signed by a certificate along with the certificate itself, binding the delegated credential to the certificate for which it is acting as "power of attorney". A supporting client indicates its support for delegated credentials by including an extension in its Client Hello.

A server that supports delegated credentials composes the TLS Certificate Verify and Certificate messages as usual, but instead of signing with the certificate's private key, it includes the certificate along with the DC, and signs with the DC's private key. Therefore, the private key of the certificate only needs to be used for the signing of the DC.

Certificates used for signing delegated credentials require a special X.509 certificate extension (currently only available at DigiCert).

Automatic Certificate Management Environment (ACME)



The Automatic Certificate Management Environment (ACME) protocol is a communications protocol for automating interactions between certificate authorities and their users' web servers, allowing the automated deployment of public key infrastructure at very low cost. It was designed by the Internet Security Research Group (ISRG) for their Let's Encrypt service.

The ACME protocol makes it possible to set up an HTTPS server and have it automatically obtain a browser-trusted certificate, without any human intervention. This is accomplished by running a certificate management agent on the web server.

The protocol, based on passing JSON-formatted messages over HTTPS, has been published as an Internet Standard in RFC 8555.

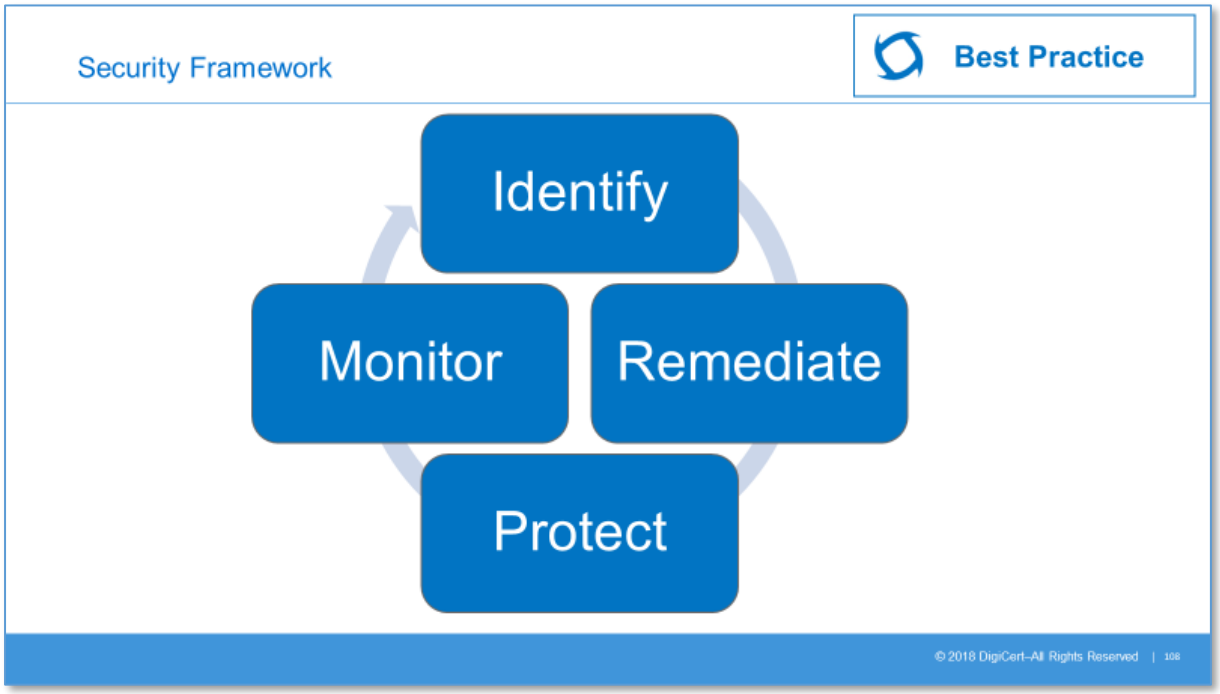
ACME v1 was released April 12, 2016. It supports issuing certificates for single domains, such as example.com or cluster.example.com.

ACME v2 was released March 13, 2018. ACME v2 is not backwards compatible with v1. Version 2 supports wildcard domains, such as *.example.com.

Using ACME, the CA verifies that the client controls the requested domain name(s) by having the ACME client perform some action(s) that can only be done with control of the domain name(s). For example, the CA might require a client requesting example.com to provision a DNS record under example.com or an HTTP resource under http://example.com.

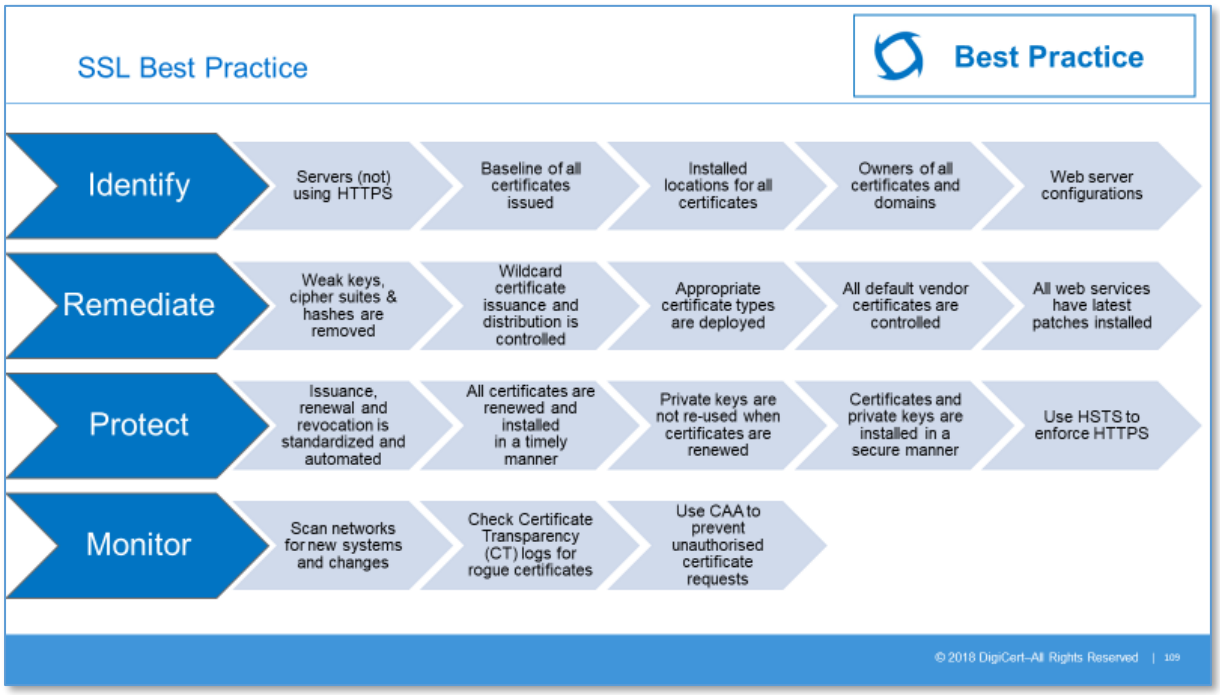
SSL/TLS Best Practice

Security



The simple security framework presented includes 4 main steps:

1. **Identify** assets;
2. **Remediate** discovered problems;
3. **Protect** assets by creating secure policies;
4. **Monitor** assets to ensure policy compliance.



Identify

Servers (not) using HTTPS

It's important to discover which web servers are (or are not) using HTTPS. With the release of the Google Chrome 68 browser, any web page not running HTTPS with a valid TLS certificate will show a 'not secure' warning in the Chrome address bar. This warning will apply to Internet-facing websites and potentially millions of corporate/private intranet sites accessed through Chrome.

Large enterprises should scan their networks and public websites to detect sites not using HTTPS and determine whether these sites should have HTTPS enabled.

Baseline of all certificates issued

SSL certificates are commonly deployed to protect both external (customer-facing) and internal applications such as financial systems, HR systems and databases.

If you don't have an up-to-date inventory of your certificate landscape, you can open yourself to security risk such as certificates expiring, weak keys and hashes.

Your inventory should provide detailed certificate information that lists the types of certificates (OV, DV, EV, private, self-signed) from all CAs as well as identifying problems with issuers, key lengths, algorithms and other certificate elements, including expiration date. Not knowing you have weak keys, algorithms and long certificate validity periods can leave you vulnerable to SSL private key compromise which in turn can lead to a "man in the middle" attack.

How do you do this? You can get a list of issued certificates from your Certificate Authorities – but have you captured everything? What about internal CAs? How about network devices with SSL certificates? For large enterprises, the best method is to use a network scanner which can detect SSL certificates.

Many customers are amazed at just how many SSL certificates they have online which nobody knew about.

Installed locations for all certificates

Just because you've issued a certificate does not mean that it has been correctly installed in the correct location, or at all! With the SSL inventory as a baseline, you should add verified certificate server locations.

You can manually track this, but for large enterprises a periodic certificate discovery scan should be used to validate that the certificate is installed in its intended location. Any missing or unknown certificates should be investigated immediately.

Remember – rogue certificates can be installed undetected if you are not verifying the correct certificate is installed at the current location. When a rogue certificate is installed it can allow encrypted traffic to leave the network perimeter without your knowledge.

Owners of all certificates and domains

If the certificate owner is not known, certificates may be renewed for no good reason, resulting in extra expense. Another common problem is where a server admin leaves the company with no handover, and the server certificate expires.

It is therefore vital to define who "owns" each certificate and define a clear process for renewals, transfer of ownership, etc.

Don't forget your domains! Ownership of public domains must also be clearly defined and up to date. A Certificate Authority cannot issue a certificate for a public domain unless it can verify domain ownership.

Web server configurations

The inventory information should also include details of the O/S and application versions. Many enterprise organizations are still vulnerable to exploits which attack specific versions of OpenSSL (like Heartbleed) because they are not aware of which systems have weakness.

Many SSL-specific attacks focus on older versions of SSL or insecure cipher suites. Significant examples include the POODLE attack on SSL v3 and the ROBOT attack on RSA Encryption. Therefore, it is also important to check the configuration of each web server and application and list which cipher suites and SSL versions are supported.

Remediate

Weak keys, cipher suites & hashes are removed

Your certificates contain public keys and signatures which could be vulnerable. Certificates with key lengths less than 2048 bits or using the MD5 or SHA1 hashing algorithms are not permitted on public web servers any more but can still be found on internal web sites. You should review any sites with weak keys and algorithms and upgrade where possible.

Perhaps more important is to review the list of SSL/TLS versions and cipher suites supported. Here are the main recommendations:

- You must disable support for SSLv2, SSLv3, and TLS 1.0 because they are outdated and vulnerable (and also to maintain PCI DSS compliance);
- You should disable TLS 1.1 if you can - there are no known security vulnerabilities, but it does not have the modern cryptographic algorithms found in TLS 1.2;
- You should enable TLS 1.2 and 1.3;
- You should disable weak ciphers (DES/3DES, RC4), and prefer modern ciphers (AES, ChaCha20), and modes (GCM).

A big problem with TLS 1.2 is that it's often not configured properly it leaves websites vulnerable to attacks. TLS 1.3 now removes obsolete and insecure features from TLS 1.2, including the following: SHA-1, RC4, DES, 3DES, AES-CBC, MD5, arbitrary Diffie-Hellman groups.

Wildcard certificate issuance and distribution is controlled

Wildcard certificates are server certificates which contain a wildcard (*) as part of the hostname. They offer a great advantage as one hostname containing a wildcard can match multiple hostnames provided they satisfy the condition.

A major concern with wildcard certificates is that if the private key and wildcard certificate are stolen at any point, the attackers can then impersonate any system in that wildcard space, potentially creating a major security breach. The danger of wildcard certificates is that they can be used on any server; even a hacker's rogue computer on the network. Common examples of using stolen wildcard keys are DNS poisoning and creating rogue wireless Access Points.

Another issue with wildcards is that if a wildcard certificate is compromised then it is necessary to revoke and reissue all copies of the certificate at all locations where it has been installed. So, the more copies of a wildcard certificate you have, the more of a headache this becomes. Of course, unless well documented, you also may not be certain that all copies have indeed been replaced. A

more secure approach is to issue each copy of a wildcard certificate with a different private key, so if one wildcard is compromised then you do not have to revoke all copies.

Wildcard certificates have their uses, but avoid using them if it means exposing the underlying keys to a much larger group of people, and especially if crossing team or department boundaries. In other words, the fewer people there are with access to the private keys, the better.

For these reasons, wildcards are not permitted in EV (extended validation) certificates. However, there is nothing wrong with using wildcards in other certificate types if well controlled and documented. For example, wildcard certificates are particularly convenient if you have a system which generates subdomains automatically.

In general, a controlled process must be put in place for issuing new wildcards to guarantee they are only installed in the intended location. Also, installing certificates on random servers without change approval can lead to unmanaged wildcards. Exporting a wildcard certificate with a private key must be closely monitored and separation of duties must be enforced.

Appropriate certificates types are deployed

The first step is to ensure that SSL certificates are installed on all web services where HTTPS is required. But which type of certificate?

Self-signed SSL certificates provide encryption but are not trusted by browsers. They should not be used because users will become accustomed to bypassing browser warnings and therefore become more susceptible to “man-in-the-middle” attacks.

Private SSL certificates can be used to protect internal systems provided the private root has been successfully propagated to all internal users of that system.

DV certificates should be used in situations where trust and credibility are less important, such as personal websites and small forums that need basic encryption for things like logins, forms or other non-transactional data. DV certificates should only be used for web-based applications that are not at risk for phishing or fraud. Avoid using them for public websites that handle sensitive data.

OV certificates should be deployed on public-facing websites dealing with less sensitive transactional data. With OV certificates, company information is displayed to users, and so they provide a certain level of trust about the company who owns the website.

EV certificates should be used on e-commerce sites and websites handling credit card and other sensitive data. EV certificates display the green address bar in most browsers. They have been shown to increase user trust, lower bounce rates and shopping cart abandonment.

All default vendor certificates are controlled

What are vendor certificates? These are the self-signed certificates used by network appliances, servers and their management systems like Drac and iLO. They all typically come with a self-signed SSL certificate pre-installed so the network resource works right out of the box for configuration.

However, it was never the intention of the vendors to have these self-signed put on a production network. These certificates are typically self-signed (untrusted) and/or expired and/or use weak keys, so will generate browser warnings which users will just click past, giving rise to the possibility of a “man-in-the-middle” attack. Also, the private key is often well known publicly and could be used to comprise a system without any knowledge.

Unfortunately, many organizations don't know about the risk. The challenge in an enterprise environment is that you may have 1000's of these.

All these default vendors certificates should be removed from the enterprise and replaced with a certificate with known trust. At a minimum a private internal CA must be used.

Replacing all these certificates in a large enterprise can be challenging. Automated tools for the installation/replacement of SSL certificates would be the key to success for these situations.

All web servers have latest patches installed

Patching operating systems and applications should be standard policy to avoid known security risks. As an example, vulnerability to Heartbleed was resolved by updating OpenSSL to a patched version (1.0.1g or later). This applies both to the web server (IIS, Apache, etc) as well as the underlying O/S (Windows, Linux, etc).

Protect

Issuance, renewal and revocation is standardized and automated

Implementing a standard process will help prevent user errors and helps implementation of separation of duties. Standardisation also enables process automation. Using automation tools further helps to reduce process errors.

Many certificates allow access to network systems or encryption and decryption with user name and password. This could allow unintended access. Certificate removal and revocation should be part of the change control in the decommissioning process for network systems and for employee off-boarding.

It is also good practice to remove private roots from user root stores when they are no longer required.

All certificates renewed and installed in a timely manner

Renew certificates yearly and more often if you can automate the process. Most sites should assume that a compromised certificate will be impossible to revoke reliably; certificates with shorter lifespans are therefore more secure in practice.

To prevent certificate outages, renew certificates 15 days prior to expiration. This practice:

- helps avoid certificate warnings for some users who don't have the correct time on their computers;
- helps avoid failed revocation checks with CAs who need extra time to propagate new certificates as valid to their OCSP responders;
- helps to ensure sufficient time for testing of certificate function. If there are any issues with a certificate install this allows time for rollback.

If you have a long change control process, then 30 days may be more appropriate.

Warning of certificate expiration should automatically be sent at regular intervals, e.g. 90 days, 60 days, 30 days, 15 days, and so on.

Private keys are not re-used when certificates are renewed

Unless keeping the same keys is important for public key pinning, you should also generate new private keys whenever you're getting a new certificate. By utilizing key rotation during certificate renewal, you limit the risk posed by a compromised key or poor key storage hygiene.

Never reuse the CSR, because re-using the CSR implies reuse of your private key as well. Controls within the O/S can be put in place, so private keys are not renewed. Audits will show how well the process is working.

Certificates and private keys are installed in a secure manner

Make sure you create your own private keys on a secure and trusted computer. Password-protect keys from the start to prevent compromise when they are stored in backup systems. Only give access to private keys as needed. When an employee with private key access leaves your company, generate a new private key.

The protection of the private key is essential and great care must be taken. Using encrypted email is one way to distribute certificates with private keys. However, the email system must expire and dispose of the emails automatically.

The systems that create the certificates (with private keys) must be well protected. Two factor authentication should be required when accessing these systems. A documented process is required when the private key is exported, moved, and stored elsewhere.

As an example, often a Dev team will stand up their own issuing CA so they can issue their own certificates at will. Software developers should never have access to an issuing CA; a formal process must be put in place for all certificate users. All certificate users must request certificates from a separate PKI team and get manager approvals with a justified business use case. By implementing separation of duties, you can enforce limited access to private keys with audit trails.

Another example is in the use of the default IIS directory. It is common knowledge that the default folder for IIS websites is found in the C:\inetpub\wwwroot. This increases the risk for breaches such as directory traversal attacks. Microsoft recommends moving the website directory to the D: drive.

Use HSTS to enforce HTTPS

For all web sites using HTTPS, use HSTS (HTTP Strict Transport Security). HSTS forces browsers and web connections to use HTTPS only.

Monitor

Scan networks for new systems and changes

Of course, networks are dynamic and constantly changing. Once you have remediated your existing systems and defined your policies, you need to monitor for new systems or any changes to existing systems. This is best achieved through regular network scans.

Scanning services are available from several vendors, including DigiCert. Ideally, they should highlight SSL security issues and certificate expirations in addition to any network changes.

Check Certificate Transparency (CT) logs for rogue certificates

Although HTTPS is an encrypted protocol, its security rests on two unencrypted protocols: BGP and DNS. Even if you have perfect HTTPS, an attacker who can subvert DNS or BGP can obtain trusted certificates for your domain and intercept all traffic between your customers and your website, exposing your business to liability and reputational damage.

BGP and DNS both have a history of subversion. In 2018, a BGP hijack of Amazon's Route 53 DNS service was used to steal cryptocurrency. In 2019, US-CERT detected a global campaign to hijack DNS infrastructure.

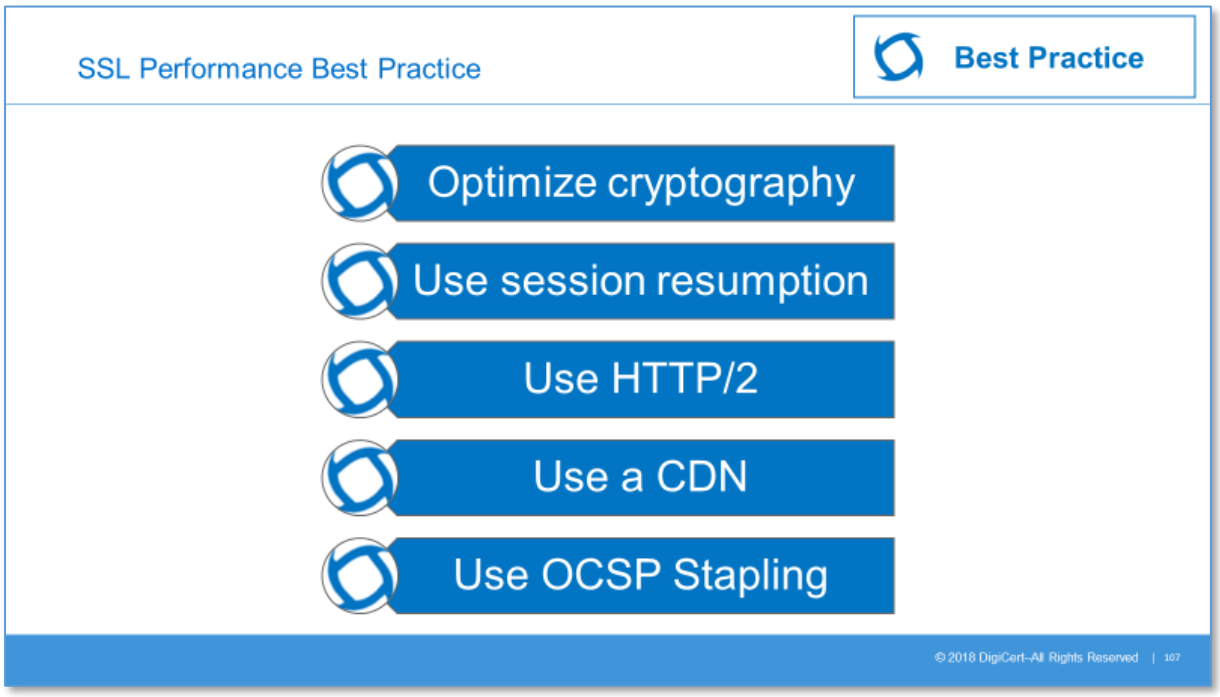
After compromising DNS or BGP, attackers must request a publicly-trusted certificate so they can intercept traffic to your site without triggering browser warnings. Using a Certificate Transparency monitor allows you to detect the issuance of an unknown certificate, so you can respond and remediate the compromise.

Use CAA to prevent unauthorised certificate requests

PKI administrators can implement controls within their environment to prohibit users from circumventing standard SSL certificate processes. A useful tool here is the DNS Certification Authority Authorization (CAA) record which can be used to specify which certificate authorities (CAs) are allowed to issue certificates for a domain.

Performance

We must also pay attention to performance; a secure service that does not satisfy performance criteria will no doubt be dropped.



Optimize cryptography

The cryptographic handshake, which is used to establish secure connections, is an operation whose cost is highly influenced by private key size. Using a key that is too short is insecure, but using a key that is too long will result in “too much” security and slow operation. For most web sites, using RSA keys stronger than 2048 bits and ECDSA keys stronger than 256 bits is a waste of CPU power and might impair user experience. Similarly, there is little benefit to increasing the strength of the ephemeral key exchange beyond 2048 bits for DHE and 256 bits for ECDHE. There are no clear benefits of using encryption above 128 bits.

Use session resumption

Session resumption is a performance-optimization technique that makes it possible to save the results of costly cryptographic operations and to reuse them for a period of time. A disabled or non-functional session resumption mechanism may introduce a significant performance penalty.

Use HTTP/2

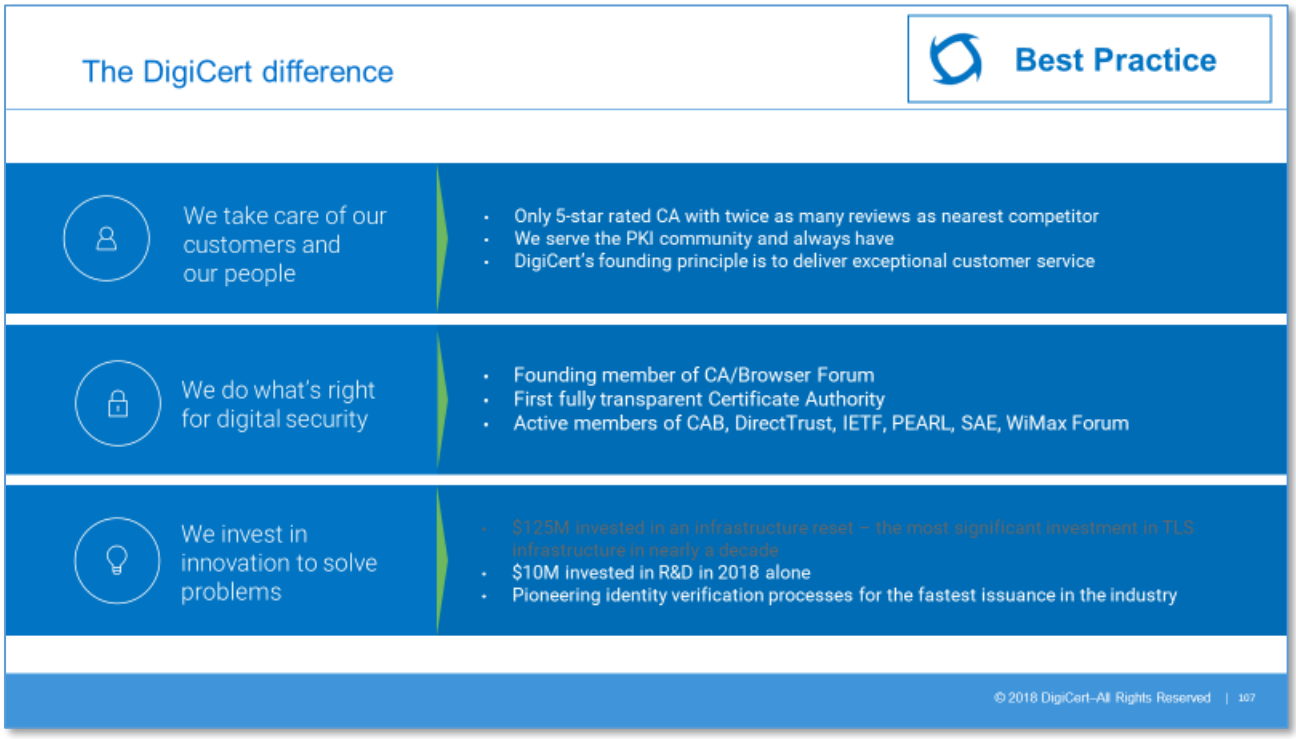
These days, TLS overhead doesn't come from CPU-hungry cryptographic operations but from network latency. HTTP/2 can reduce network latency by techniques including multiplexing, concurrency, header compression, etc.

Use a CDN

A TLS handshake, which can start only after the TCP handshake completes, requires a further exchange of packets and is more expensive the further away you are from the server. A CDN (Content Delivery Network) can provide optimaization by caching content and connecting users to their nearest POP (point of presencse).

Use OCSP Stapling

Online Certificate Status Protocol (OCSP) is a protocol that can be used to check certificate revocation status in real time. OCSP Stapling is a TLS protocol feature that enables OCSP stauts to be embedded in the TLS handshake with the server that uses the certificate. As a result, the client does not need to contact OCSP servers for out-of-band validation and the overall TLS connection time is significantly reduced. OCSP stapling is an important optimization technique, but you should be aware that not all web servers provide solid OCSP stapling implementations. Combined with a CA that has a slow or unreliable OCSP responder, such web servers might create performance issues. For best results, simulate failure conditions to see if they might impact your availability.



One final best practice: Select a Certification Authority (CA) that is reliable and serious about their certificate business and security. Consider the following criteria when selecting your CA:

- **Security posture:** All CAs undergo regular audits, but some are more serious about security than others. Figuring out which ones are better in this respect is not easy, but one option is to examine their security history, and, more important, how they reacted to compromises and if they learned from their mistakes.

- **Business focus:** CAs whose activities constitute a substantial part of their business have everything to lose if something goes terribly wrong, and they probably won't neglect their certificate division by chasing potentially more lucrative opportunities elsewhere.
- **Services offered:** At a minimum, your selected CA should provide support for both Certificate Revocation List (CRL) and Online Certificate Status Protocol (OCSP) revocation methods, with rock-solid network availability and performance. You should also have a choice of public key algorithm. Most web sites use RSA today, but ECDSA may become important in the future because of its performance advantages.
- **Certificate management options:** If you need a large number of certificates and operate in a complex environment, choose a CA that will give you good tools to manage them.
- **Support:** Choose a CA that will give you good support if and when you need it.

Case Study Exercise

About DigiBank

DigiBank is a small regional bank servicing customer in Lehi, UT and the Salt Lake City UT area. They have about 10,000 employees, and over 100 branch locations. They maintain an on-premises data center in Lehi, and have a backup data center in MT View, CA for disaster recovery and operational continuity.

DigiBank maintains about 5,000 on-premises servers and manages a couple hundred in the cloud using AWS and other providers. They have issued over 14,000 TLS/SSL certificates for various purposes. In addition, every user maintains two certificates for email signing/encryption and authentication.

DigiBank has recently constructed a new internal PKI environment based on Microsoft Certificate Services to support their move to SHA256 based certificates. In the past, DigiBank has experienced several outages due to certificate expiration, but fortunately none of these have caused significant business impact.

The CISO team recognizes the potential for more significant operational and security impact related to certificates due to possible weak keys, vendor certs and self-signed certificates and has therefore made PKI and certificate management a top priority for the security team.

DigiBank's initial goals include identification of all existing SHA1 certificates issued by the legacy CA, and a process for actively pursuing the reissuance using the new SHA256 PKI. Ensuring proper notifications and escalations to prevent expiration related outages is a primary objective as well. Also weak keys, vendor certs and self-signed certificates need to be phased out.

There are various teams that make the request for certificates. For example DigiBank has firewall/load balancer, Linux, Windows, Network, Database and Web teams.

Currently all requests for certificates are being manually fulfilled by the InfoSec Admin team. DigiBank recognizes the inability for this model to scale and has set the goal for the InfoSec Admin team to have 80% or more of all certificates requested performed directly by the requesting teams within 3 months' time.

DigiBank Q&A

The following questions were answered by various departments at DigiBank:

Does DigiBank have **a Baseline of all certificates** issued by the enterprise?
We rely on a Microsoft export of our internal CA and an export of the certificates from our Public CA. Our vulnerability scanner can detect a certificate but cannot retrieve and store the necessary metadata to manage the cert. or perform any kind of meaningful reporting or action.

Does DigiBank know the **Installed locations** for all TLS/SSL **server-side** certificates?
We know most of our servers were issued certificates from the Microsoft CA export and some locations from our vulnerability scans. But we currently don't have an accurate way of reporting to be proactive in searching for certificate anomalies.

How are **new anomalous certs** detected and remediated inside the enterprise?
We don't know when an anomalous certificate is installed. Our vulnerability scans may detect it be don't have any reporting to know if the certificate is rogue and we don't have process in place to remediate.

How are **weak keys & hashes** removed from environment?
We remove weak keys and hashes manually when they are reported by staff. However, there is a culture by the operations team is if it works don't fix it. There is real fear of replacing certs as this has resulted in outages.

How do you prevent **Private keys being re-used** when certificates are renewed?
Currently any admin can accidently re-use keys when issuing a new certificate of the Microsoft CA. When this happens we have no way of knowing if the private key has been changed or re-used. Only two admins can issue public certs so this easy to manage via the Vendors portal.

Is **wildcard certificate issuance** and distribution controlled?
We started out with only known wildcard locations and we thought it was well documented. However, as admins have come and gone, wildcards certs seem to be popping up on load-balancers and servers without authorization. This usually happens when a certificate is being renewed and for whatever reason does not work. In a panic we install a wildcard certs to prevent an outage.

Are **Validity periods** for **public-facing** certificates reduced to ≤ 1 year?
No, 3 years is what we purchase with the public CA. DigiCert has recently reduced this to 2 years validity periods per the CAB forum requirements.

Are **all default vendor certificates** controlled?
No, we have self-signed Vendor certs with our network appliances. 1000s of IDACS, ILOs, routers, switches. It's a mess we have not even considered messing with.

Are **owners of all certificates** identified and assigned to groups?
Yes. Each group put in their request for certificates to our PKI team. Our PKI team is overwork with renewals. Many internal certs have been issued for 5 years so the PKI admin won't have to deal with the renewal during their tenure. We have a lot of turnover with our PKI admins.

Are **Issuance & renewal process** standardized and streamlined for all CA's?
We have a process in place for the Microsoft CA. But it's hard for us to enforce and is easily bypassed. Public CA's have their portal so it easy to control the issuance and renewal process.

Are all certificates renewed and installed **at least 15 days before expiry?**
We try to renew before 15 days, but we have outages. It's hard to keep up. We get surprised all the time.

Is **Certificate removal/revocation** addressed in decommissioning process?
None currently. We had a terminated employee use his cert. on his mobile device that only used certificate authentication. We need help with this process.

Are all certificate and private keys **monitored for changes?**
We currently have no way to report a change when private keys change. We only rotate keys when we renew a certificate.

Do Certificate and key generation processes address **separation of duties?**
We have a process but due to lack of staff this gets bypassed.

Are all private keys **securely generated** and stored?
The Microsoft CA generates and stores the private keys to a secure internal base.

Are **Access to private keys** by individuals minimized and auditable?
Any Admin can assess our Microsoft PKI and export a certificate with a private key.

Are Certificates and private keys **installed in secure manner?**

We send our certs with a private key (.pfx) via secure email to end users for remote VPN access that require a certificate.

Student Exercise

Working in pairs or larger groups, discuss the DigiBank situation:

- How compliant are they with best practice? Rate them on a scale 1-10.
- What are the top 5 things you would recommend for DigiBank to be more compliant with best practice?

Prepare a short presentation of your findings and recommendations to present back to the class.

Appendix A: SSL& TLS History

SSL versions 1, 2, and 3

The SSL protocol was originally developed by Netscape. Version 1.0 was never publicly released; version 2.0 was released in February 1995 but "contained a number of security flaws which ultimately led to the design of SSL version 3.0". (Rescorla 2001) SSL version 3.0 was released in 1996.

TLS version 1.0

TLS 1.0 was first defined in RFC 2246 in January 1999 as an upgrade to SSL Version 3.0. As stated in the RFC, "the differences between this protocol and SSL 3.0 are not dramatic, but they are significant enough that TLS 1.0 and SSL 3.0 do not interoperate." TLS 1.0 does include a means by which a TLS implementation can downgrade the connection to SSL 3.0.

TLS version 1.1

TLS 1.1 was defined in RFC 4346 in April 2006. It is an update from TLS version 1.0. Significant differences in this version include:

- Added protection against Cipher block chaining (CBC) attacks.
- The implicit Initialization Vector (IV) was replaced with an explicit IV.
- Change in handling of padding errors.
- Support for IANA registration of parameters.

TLS version 1.2

TLS 1.2 was defined in RFC 5246 in August 2008. It is based on the earlier TLS 1.1 specification. Major differences include:

- The MD5/SHA-1 combination in the pseudorandom function (PRF) was replaced with SHA-256, with an option to use cipher-suite specified PRFs.
- The MD5/SHA-1 combination in the Finished message hash was replaced with SHA-256, with an option to use cipher-suite specific hash algorithms.
- The MD5/SHA-1 combination in the digitally-signed element was replaced with a single hash negotiated during handshake, defaults to SHA-1.
- Enhancement in the client's and server's ability to specify which hash and signature algorithms they will accept.
- Expansion of support for authenticated encryption ciphers, used mainly for Galois/Counter Mode (GCM) and CCM mode of AES encryption.
- TLS Extensions definition and Advanced Encryption Standard (AES) CipherSuites were added.

TLS version 1.3

TLS 1.3 was defined in RFC 8446 in August 2018. Main changes include:

- Removing support for weak and lesser-used named elliptic curves and cryptographic hash functions
- Requiring digital signatures even when a previous configuration is used
- Integrating HKDF and the semi-ephemeral DH proposal
- Replacing resumption with PSK and tickets

- Supporting 1-RTT handshakes and initial support for 0-RTT (round-trip delay time)
- Dropping support for many insecure or obsolete features including compression, renegotiation, non-AEAD ciphers, static RSA and static DH key exchange, custom DHE groups, point format negotiation, Change Cipher Spec protocol, Hello message UNIX time, and the length field AD input to AEAD ciphers
- Prohibiting SSL or RC4 negotiation for backwards compatibility
- Integrating use of session hash
- Deprecating use of the record layer version number and freezing the number for improved backwards compatibility
- Moving some security-related algorithm details from an appendix to the specification and relegating ClientKeyShare to an appendix
- Addition of the ChaCha20 stream cipher with the Poly1305 message authentication code
- Addition of the Ed25519 and Ed448 digital signature algorithms
- Addition of the x25519 and x448 key exchange protocols

Appendix B: Useful Links

SSL Standards

TLS 1.2: https://tools.ietf.org/html/rfc5246

TLS 1.3: https://tools.ietf.org/html/rfc8446

CA/Browser Forum Baseline Requirements Version 1.5.7 April 29, 2018: https://cabforum.org/baseline-requirements-documents/

CA/Browser Forum Guidelines for the Issuance and Management of Extended Validation Certificates v1.6.8: https://cabforum.org/extended-validation/

Validation

WHOIS, GDPR and Domain Validation: https://www.digicert.com/blog/note-on-whois-gdpr-and-domain-validation/

Other Protocols

SNI: https://tools.ietf.org/html/rfc6066#section-3

AIA: https://www.thesslstore.com/blog/aia-fetching/

OCSP: https://tools.ietf.org/html/rfc6960

SSL Risks & Vulnerabilities

TLS vulnerability and attacks: https://www.acunetix.com/blog/articles/tls-vulnerabilities-attacks-final-part/

BEAST attack: http://resources.infosecinstitute.com/ssl-attacks/#gref

ROBOT attack: https://thehackernews.com/2017/12/bleichenbacher-robot-rsa.html

https://robotattack.org/

CA breaches: http://blog.isc2.org/isc2_blog/2012/04/test.html

https://www.securityweek.com/lessons-learned-diginotar-comodo-and-rsa-breaches

https://spectrum.ieee.org/riskfactor/telecom/security/diginotar-certificate-authority-breach-crashes-egovernment-in-the-netherlands

OpenSSL Heartbeat Explained---Stealing Credit Cards: https://www.youtube.com/watch?v=hTK0pywfmDE

The POODLE bug! SSL vulnerability explained: https://www.youtube.com/watch?v=C8ks8WLoZto

Man in the Middle explained: https://www.youtube.com/watch?v=gNhyjPxuy5w

Exploits weak keys: https://www.youtube.com/watch?v=Pe_B--xdQdE

SHA-1 Exploit Explained: https://www.youtube.com/watch?v=2UuSxFaoxhI

Industry Trends

Certificate Transparency:

https://www.certificate-transparency.org/

https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Expect-CT

https://support.apple.com/en-gb/HT205280

CAA:

https://tools.ietf.org/html/rfc6844

https://www.digicert.com/dns-caa-rr-check.htm

HSTS: https://tools.ietf.org/html/rfc6797

HTTP/2: https://tools.ietf.org/html/rfc7540

US Government Compliance Guide: https://https.cio.gov/guide/#compliance-and-best-practice-checklist

"Always-on" SSL:
https://otalliance.org/resources/always-ssl-aossl
https://casecurity.org/2016/09/30/always-on-ssl/
https://www.digicert.com/always-on-ssl.htm

SXG:

https://blog.hubspot.com/marketing/google-amp

https://medium.com/oyotech/implementing-signed-exchange-for-better-amp-urls-38abd64c6766

Delegated Credentials:

https://blog.cloudflare.com/keyless-delegation/

https://engineering.fb.com/security/delegated-credentials/

Best Practice

CA Security Council: https://casecurity.org/2014/02/26/pros-and-cons-of-single-domain-multi-domain-and-wildcard-certificates/

SANS Institute:

https://www.sans.org/reading-room/whitepapers/analyst/critical-security-controls-guidelines-ssl-tls-management-35995

https://www.sans.org/reading-room/whitepapers/authentication/dangers-weak-hashes-34412

https://www.sans.edu/cyber-research/security-laboratory/article/it-separation-duties

NIST:

Wildcard use: Page 15: https://csrc.nist.gov/CSRC/media/Projects/Computer-Security-Objects-Register/documents/ACES-CP-v3-2_signed_05122017.pdf

Private Key: Protection & Assurance Requirements Page 75:
https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57pt1r4.pdf

OWASP:

https://www.owasp.org/index.php/Key_Management_Cheat_Sheet#Key_Rotation

CA/Browser Forum:

CERTIFICATE LIFE-CYCLE OPERATIONAL REQUIREMENTS:

https://cabforum.org/wp-content/uploads/CA-Browser-Forum-BR-1.5.3.pdf

Other:

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

Protecting critical infrastructure from cyber threats information technology systems: http://www.dla.mil/AboutDLA/News/NewsArticleView/Article/1324406/protecting-critical-infrastructure-from-cyber-threats-information-technology-sy/

Detection of Rogue Certificates:

https://www.researchgate.net/publication/316940647_Detection_of_Rogue_Certificates_from_Trusted_Certificate_Authorities_Using_Deep_Neural_Networks