Strong SSL Configuration & Compatibility Report

Request for comments
- DRAFT

Thierry ZOLLER
Principal Security Consultant
contact@g-sec.lu
http://www.g-sec.lu

G-SEC™ is a **vendor independent** Luxemburgish led security consulting group that offers IT Security consulting services on an organizational and technical level. Our work has been featured in New York Times, eWeek, ct', SAT1, Washington Post and at conferences ranging from Hack.lu to Cansecwest.

Table of Contents

Executive Summary

This report gives general recommendations as to how to configure SSL/TLS in order to provide state of the art authentication and encryption support. The options offered by SSL engines grew from the early days since Netscape developed SSL2.0 to the introduction of TLS. Furthermore servers and clients offer a different set of available options and finding the middle ground has proven difficult.

We will also explain the various vulnerabilities present in SSL/TLS, how to mitigate them and provide guidance as to support a wide variety of Browsers and still offer state of the art security. For this purpose this paper comes with a toolset which consists of:

- SSL Harden (beta) – Allows users of Windows 2000, XP, Vista, 7 and particularly administrators of Windows Server 2003 & 2008 to configure SSL/TLS support. Administrators can manually edit and backup the SSL configuration and set PCI-DSS compliant SSL rules with a click of a button. To ease administration SSL Harden allows to read and write the settings remotely via RPC/SMB.

- SSL Audit (alpha) - is a remote SSL audit tool able scan for SSL/TLS support against remote servers; it does so by using its own small parsing engine and does not rely on OpenSSL or other engines.

The paper has been written with e-banking applications in mind and solely comments about the underlying SSL/TLS security protocol; no other security mechanism such as (TAN, mTAN, iTAN, OTP..) or other mechanisms have been taken into account.

The information is believed to be correct at the time of writing, if you believe the information displayed within this paper is wrong please contact contact@g-sec.lu

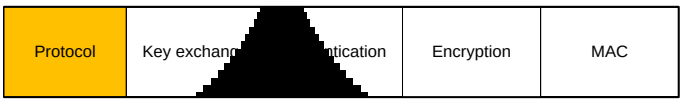
Revisions

Version	*Date*	*Annotations*	*Author*
0.8	07.12.2009	Initial draft	Thierry ZOLLER
0.85	09.12.2009	Added recommendations, Added BSI, NIST, FSIA recommendations	Thierry ZOLLER
0.9	09.12.2009	Added Browser support Added Server support	Thierry ZOLLER
0.95	18.12.2009	Executive summary	Thierry ZOLLER

SSL/TLS

In order to securely transport data from one endpoint to another SSL and TLS protocols are used as they provide data confidentiality and data integrity. TLS was designed to offer a flexible and secure protocol that is able to interoperate with any service or application, furthermore TLS provides cryptographic support that SSL could not offer.

SSL/TLS Protocol versions



SSLv2

SSL version 2 was developed by Netscape in 1996 and is 13 years old; it is vulnerable to various attacks and should not be supported. Several E-banking sites do although internet browsers like Internet Explorer 7 (2006), Firefox 2 (2005) and Opera 9 (2006) **do no longer support SSLv2**.

Users should not be encouraged to use older browsers as they suffer from other vulnerabilities that put them and their banking information at risk. Should another requirement such as third party code require SSLv2 for an e-banking platform it needs to be upgraded to TLS, as it is vulnerable to several known attacks.

Should you absolutely need to conform to foreign regulations (china and others) we recommend relocating these customers to a separated banking server/system. They pose a risk for other e-banking users. (SSLv2 does not support perfect forward secrecy)

The SSLv2 protocol suffers from

- Re-usage of key material (message authentication and encryption) thus, in case of EXPORT ciphers, also unnecessarily weakening the MAC (not required by export restrictions)
- Ciphers marked as “Export” have an arbitrary small key size and can be cracked easily with today’s hardware.
- weak MAC construction and supports only MD5 hash function
- padding length field is unauthenticated [1]
- Downgrade attack – an attacker may downgrade the encryption to the lowest available and after doing so crack the keys.
- Truncation attacks – The attacker may reset the TCP connection and as such

[1] Analysis of the SSL 3.0 Protocol - David Wagner et al

Differences between SSLv3 and SSLv2

- Key material is no longer reused in both Message authentication and encryption making suites marked as EXPORT „stronger“.
- MAC contruction enhanced and support for SHA1 added
- SSLv3 adds protection of the Handshake, server-side can detect downgrade attacks
- SSLv3 adds support for a closure alert

Differences between TLS v1and SSLv3

- Expansion of cryptographic keys from the initially exchanged secret was improved
- MAC construction mechanism modified into an HMAC
- Mandatory support for Diffie-Hellman key exchange, the Digital Signature Standard, and Triple-DES encryption

Differences between TLS v1.1 and TLS v1 [2]

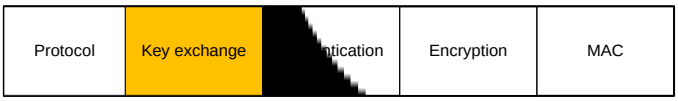
- The implicit Initialization Vector (IV) is replaced with an explicit IV to protect against CBC attacks[3]
- Handling of padding errors is changed to use the bad_record_mac
- Alert rather than the decryption_failed alert to protect against CBC attacks
- IANA registries are defined for protocol parameters.
- Premature closes no longer cause a session to be nonresumable.
- Additional informational notes were added for various new attacks on TLS

Differences between TLSv1.2 and TLSv1.1 [4]

- SHA-256 is the default digest method
- Several new cipher suites use SHA-256
- It has better ways to negotiate what signature algorithms the client supports
- Alerts are mandatory now be sent in many cases
- After a certificate_request, if no certificates are available, clients now MUST send an empty certificate list
- TLS_RSA_WITH_AES_128_CBC_SHA is now the mandatory to implement cipher suite
- Added HMAC-SHA256 cipher suites
- Removed IDEA and DES cipher suites, they are now deprecated.
- Support for the SSLv2 backward-compatible is now optional only.

[2] http://www.ietf.org/rfc/rfc4346.txt
[3] http://www.openssl.org/~bodo/tls-cbc.txt
[4] http://www.ietf.org/rfc/rfc5246.txt

Protocol Key exchange



The key exchange is used to generate a pre_master_secret known to the client and the server but not to somebody in the middle of the connection (Attacker). The pre_master_secret is then used to generate the master_secret which is used to generate the certificate verify and finished messages, encryption keys, and MAC secrets.

RSA

With RSA, key exchange and server authentication are combined. The public key may be either contained in the server's certificate or may be a temporary RSA key sent in a server key exchange message, old signatures and temporary keys cannot be replayed.

DH

DH stands for Diffie Hellman, when using DH the server supplies a certificate containing a fixed Diffie-Hellman parameter. Temporary parameters are hashed and signed to ensure that attackers cannot replay parameters. The client then verifies the certificate and signature to ensure that the parameters belong to the actual server. When using DH the client and server will generate the same pre_master_secret every time.

DHE

DHE stands for Ephemeral Diffie Hellmann, the server supplies a certificate containing **temporary Diffie-Hellman parameter signed with the servers RSA or DSS certificate**. This has the effect that it offers perfect forward secrecy. This means that even if you have compromised/broken/stolen the server private key that **you cannot decrypt past captured traffic.**

For this case DHE is the recommended key exchange protocol by G-SEC, if for monitoring reasons decryption needs to be done we recommend to write the Diffie Hellmann parameters to a database for every new session. That way you can decrypt past traffic if you have access to the server side database.

ADH

ADH stands for Anonymous Diffie Hellmann and allows completely anonymous connections, the server and client public parameters are contained in the corresponding exchange messages. Passive man-in-the-middle attacker should not be able to find the Diffie-Hellman result (i.e. the pre_master_secret), **however this method of key exchange is vulnerable to active man-in-the-middle attacks**.

ECDHE

ECDHE (or EECDH in Openssl 1.0) is DHE combined with elliptic key cryptography.

Authentication

TLS supports three authentication modes: authentication of server and client (through server and client certificate), server only authentication and anonymous connections. The algorithms available are:

No authentication

No authentication

RSA

The algorithm used to sign the certificate is RSA[5] [6]

DSS

The digital signature standard is used to sign the certificate

ECDSA

ECDSA stands for Elliptic Curve Digital Signature Algorithm; it is a variant of the Digital Signature algorithm that uses Elliptic Curve cryptography.

KRB5[7]

Kerberos credentials are used to achieve mutual authentication and to establish a master secret which is subsequently used to secure client-server communication.

PSK

Authentication takes place pre-shared keys, these symmetric keys are known to both parties prior to authenticating.

[5] http://en.wikipedia.org/wiki/RSA
[6] http://www.di-mgt.com.au/rsa_alg.html
[7] http://www.ietf.org/rfc/rfc2712.txt

Encryption

Encryption serves the purpose to transform plaintext into unreadable data through usage of an algorithm.

NULL

No encryption will take place, this is for example useful when you want to ensure the authenticity of the data

AES[8]

The Advanced Encryption Standard, previously known as Rjindael, was the winner of the NIST competition as it regarded as state of the art encryption. AES offers key sizes from 128, 192 to 256 bits of size

CAMELLIA[9]

Developed by Mitsubishi and NTT is available under a royalty free license and according to sources has been "has been evaluated favorably by several organisations, including the European Union's NESSIE project (a selected algorithm), and the Japanese CRYPTREC project (a recommended algorithm)"

RC4 / RC2

RC4 is a Stream cipher invented by Ron Rivest and was closed source until the release of the source code in 1994 to cypherpunks mailing list. There were several attacks that have been uncovered against RC4, particularly as used within WEP. RC2 is a block cipher invented by Ron Rivest in 1996 the source code was leaked to the sci.crypt usenet group. RC2 is vulnerable to several attacks.

IDEA[10]

The International Data Encryption Algorithm is a block cipher invented by James Massey , It is still considered secure however it is patented and slower than modern ciphers. The patent will expire in 2011.

3DES

Triple-DES was created when DES was found to be vulnerable due to a key size being too small, it uses the e Data Encryption Standard cipher algorithm three times over each block.

DES

The history of DES is interesting as it was believed that the NSA tampered with the s-boxes, Wikipedia has a good summary. DES is weak and should no longer be used.

[8] http://en.wikipedia.org/wiki/Advanced_Encryption_Standard
[9] http://en.wikipedia.org/wiki/Camellia_%28cipher%29
[10] http://en.wikipedia.org/wiki/International_Data_Encryption_Algorithm

Minimum industry Encryption and Key length recommendations

This summary does not take into account the arrival of quantum computing, large quantum computers able to crack large keys are foreseen for 2014 by the ARDA and 2018 by Prof Lloyd[11] . Shors' algorithm could then be used to break the RSA key sizes presented here below.

Recommended Asymmetric key length[12]

Year	*Window*	*BSI*[13]	*NIST*[14]	*Lenstra*[15]	*FNISA*[16]
Until 2009	Minimum Recommended	1536 2048	1024	1114	1536
Until 2010	Minimum Recommended	1728 2048	1024	1152	1536
Until 2012	Minimum Recommended	1976 2048	2048	1229	2048
Until 2020	Minimum	2048	2048	1568	4096

Recommended Symmetric key length

Year	*Window*	*BSI*	*NIST*	*Lenstra*	*FNISA*
Until 2009	Minimum	-	80	74	80
Until 2010	Minimum	-	80	75	80
Until 2012	Minimum	-	112	76	100
Until 2020	Minimum	-	112	82	100

Recommended Hashing algorithm and size

Year	*Window*	*BSI*	*NIST*	*Lenstra*
Until 2009	-	80	148	160 minimum
Until 2010	-	224	150	160 minimum
Until 2012	SHA-224, SHA-256 SHA-384, SHA-512	224	152	256 minimum (SHA)
Until 2020	-	224	163	256 minimum (SHA)

[11] http://synaptic-labs.com/ecosystem/context-qc-relevant-today.html
[12] http://www.rsa.com/rsalabs/node.asp?id=2264
[13] https://www.bsi.bund.de/cae/servlet/contentblob/476754/publicationFile/31104/BSI_Final_07_pdf.pdf
[14] http://csrc.nist.gov/groups/ST/toolkit/key_management.html
[15] http://people.epfl.ch/arjen.lenstra
[16] http://www.ssi.gouv.fr/site_article76.html

G-SEC recommendations and best practices

This section gives advice on how to securely configure your SSL/TLS service and in particularly which Encryption, Authentication, Key exchange settings to use. In order to do so we collected SSL support information from all modern Browsers and servers with additional help from Ivan Ristic[17] (SSL Labs).

[17] http://blog.ivanristic.com/2009/07/examples-of-the-information-collected-from-ssl-handshakes.html

TLS / SSL Browser Compatibility overview

In order to assess the SSL/TLS support of modern Internet browsers we had to take a look at the SSL engines they use. Netscape uses the NSS engine, IE5,6,7,8 and Safari use Schannel, Opera and Safari for Mac uses custom SSL engines. *As this collection and analysis took quite some time, we would appreciate a heads-up if you use this information*.

Browser protocol support (Default)

There is no reason to continue supporting SSLv2 - Offering SSLv2 opens you to liability should a transaction be compromised.

Protocol	*Firefox 3* [1]	*XP/W2k/2003* [2]	*Vista & 7* [2]	*Opera 10*	*Safari 4* [3]
SSLv2	No	No	No	No	No
SSLv3	Yes	Yes	Yes	Yes	Yes
TLS 1.0	Yes	Yes	Yes	Yes	Yes
TLS 1.1	No	No	Yes	Yes	No
TLS 1.2	No	No	Yes	Yes	No

Browser Key exchange algorithms support

We recommend using ephemeral Diffie Hellmann paired with either RSA or DSS as signature

Algorithm	*Firefox 3* [1]	*XP/W2k/2003* [2]	*Vista & 7* [2]	*Opera 10*	*Safari 4* [3]
RSA	Yes	Yes	Yes	Yes	Yes
DHE-RSA	Yes	No	Yes	Yes	Yes
DHE-DSS	Yes	Yes	Yes	Yes	Yes
ECC	Yes	No	Yes	No	No

1 Windows, Linux, MACOSX | 2 Internet Explorer 7 & Internet Explorer 8 & Safari 4 (use schannel.dll – provided by Windows) | 3 MacOSX

RSA

RSA public-key cryptosystem is an asymmetric encryption method and (public-key cryptography) it can be used for signatures as well as encryption. In SSL/TLS RSA is used during key exchange (handshake). RSA bases its security on the length of the modulus that must be factored. The bigger the modulus the harder it is to break the algorithm.

Browser supported RSA key size, DH and SRP [18]

These are the key sizes that are supported by major Browsers, there is no client side restriction to use 1024 bit instead of 2048, and additionally 1024 bit are considered weak by today's standards.

RSA Modulus	*Firefox 3* [1]	*XP/W2k/2003* [2]	*Vista & 7* [2]	*Opera 10*	*Safari 4* [3]
1024	Yes	Yes	Yes	Yes	Yes
2048	Yes	Yes	Yes	Yes	Yes
4096	Yes	Yes	Yes	Yes	unkn

Browser supported Ciphers [19]

In order for this list to stay focused on best practices we display modern or strong ciphers only.

Cipher	*Size*	*Firefox 3* [1]	*XP/W2k/2003* [2]	*Vista & 7* [2]	*Opera 10*	*Safari 4* [3]
AES	128	Yes	No [4]	Yes	Yes	Yes
AES	256	Yes	No [4]	Yes	Yes	Yes
RC4	128	Yes	Yes	Yes	Yes	Yes
Camellia	128	Yes	No	No	No	No
Camellia	256	Yes	No	No	No	No
3DES	168	Yes	Yes	Yes	Yes	Yes

1 Windows, Linux, MACOSX | 2 Internet Explorer 7 & Internet Explorer 8 & Safari 4 (use schannel.dll – provided by Windows) | 3 MacOSX | 4 Support for AES can be added through a Hotfix

[18] http://msdn.microsoft.com/en-us/library/bb931357%28VS.85%29.aspx
[19] With heavy support from SSLLAB (Ivan Ristic)

ECC

Elliptic curve cryptography bases on a discrete logarithm problem, ECC needs less key size to achieve the same strength then RSA, as an example, an ECC 160-bit field offers the same resistance as an 1024-bit RSA modulus. This allows for smaller keys and offer improved performance. Unfortunately ECC is not widely supported in Browser as of yet, but certainly will be in the future.

ECDH support

ECDH	*Firefox 3* [1]	*XP/W2k/2003* [2]	*Vista & 7* [2]	*Opera 10*	*Safari 4* [3]
P-256	Yes	No	Yes	No	No
P-512	Yes	No	Yes	No	No

ECDSA support

ECDSA	*Firefox 3* [1]	*XP/W2k/2003* [2]	*Vista & 7* [2]	*Opera 10*	*Safari 4* [3]
P-256	Yes	No	Yes	No	No
P-512	Yes	No	Yes	No	No

1 Windows, Linux, MACOSX | 2 Internet Explorer 7 & Internet Explorer 8 & Safari 4 (use schannel.dll – provided by Windows) | 3 MacOSX

TLS / SSL Server Compatibility overview

Server protocol support

This matrix shows the protocol support of modern web servers - There is no reason to continue supporting SSLv2 - Offering SSLv2 opens you to liability should a transaction be compromised.

Protocol	*IIS6* [1]	*IIS7* [2]	*IIS7.5* [3]	*Openssl*	*GnuTls*	*JSSE 1.4.2* [4]	*NSS* [5]
SSLv2	Yes	Yes	Yes	Yes	No	Yes	Yes
SSLv3	Yes	Yes	Yes	Yes		Yes	Yes
TLS 1.0	Yes	Yes	Yes	Yes	Yes	Yes	Yes
TLS 1.1	No	Yes	Yes	No	Yes	No	No
TLS 1.2	No	No	Yes	No	Yes	No	No

Server key exchange algorithms support

We recommend offering ephemeral Diffie Hellmann paired with either RSA or DSS as signature

Algorithm	*IIS6* [1]	*IIS7* [2]	*IIS7.5* [3]	*Openssl*	*GnuTls*	*JSSE 1.4.2* [4]	*NSS*[20]
RSA	Yes	Yes	Yes	Yes	Yes	Yes	Yes
DHE-RSA	No	Yes	Yes	Yes	Yes	Yes	Yes
DHE-DSS	Yes	Yes	Yes	Yes	Yes	Yes	Yes
ECC	No	Yes	Yes	Yes[21]	Yes	No	Yes

1 Windows 2003 | 2 Windows 2008 | 3 Windows 2008 R2 | 4 Tomcat | 5 Network Security Services[22] (Apache,Redhat,Sun Java Enterprise..)

[20] http://www.mozilla.org/projects/security/pki/nss/nss-3.11/nss-3.11-algorithms.html
[21] https://issues.apache.org/bugzilla/show_bug.cgi?id=40132
[22] https://developer.mozilla.org/en/Overview_of_NSS#Interoperability_and_Open_Standards

RSA

RSA public-key cryptosystem is an asymmetric encryption method (public-key cryptography), it can be used for signing as well as encryption. In SSL/TLS RSA is used during key exchange (handshake). RSA bases its security on the length of the modulus that must be factored. The bigger the modulus the harder it is to break the algorithm.

Server RSA key size, DH and SRP prime support[23]

This list the key sizes that are supported by Major Web servers, there is no server side restriction to use 1024 bit instead of 2048.

RSA Modulus	*IIS6* [1]	*IIS7* [2]	*IIS7.5* [3]	*Openssl*	*GnuTls*	*JSSE 1.4.2*[4]	*NSS* [24]
1024	Yes	Yes	Yes	Yes	Yes	Yes	Yes
2048	Yes	Yes	Yes	Yes	Yes	Yes	Yes
4096	Yes	Yes	Yes	Yes	Yes	No	Yes

Server Cipher support[25]

In order for this list to stay focused on best practices we display modern or strong ciphers only.

Cipher	*Size*	*IIS6* [1]	*IIS7* [2]	*IIS7.5* [3]	*Openssl*	*GnuTls*[26]	*JSSE 1.4.2*[4]	*NSS* [27]
AES	128	No	Yes	Yes	Yes	Yes	Yes	Yes
AES	256	No	Yes	Yes	Yes	Yes	Yes	Yes
RC4	128	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Camellia	128	No	No	No	Yes	Yes	No	Yes
Camellia	256	No	No	No	Yes	Yes	No	Yes
3DES	156	Yes	Yes	Yes	Yes	Yes	Yes	Yes

1 Windows 2003 | 2 Windows 2008 | 3 Windows 2008 R2 | 4 Tomcat | 5 Network Security Services[28] (Apache,Redhat,Sun Java Enterprise..)

[23] http://msdn.microsoft.com/en-us/library/bb931357%28VS.85%29.aspx
[24] http://www.ibm.com/developerworks/websphere/techjournal/0612_birk/0612_birk.html
[25] With heavy support from SSLLAB (Ivan Ristic)
[26] http://www.gnu.org/software/gnutls/comparison.html
[27] http://www.ibm.com/developerworks/websphere/techjournal/0612_birk/0612_birk.html
[28] https://developer.mozilla.org/en/Overview_of_NSS#Interoperability_and_Open_Standards

Recommend SSL configuration

Taking into account the previous client and server compatibility matrixes it is apparent that the best setup to use has changed over the years. Protocols have been enhanced and weaknesses patched and encryption strengthened.

Apache & IIS7/ IIS7.5 recommended SSL configuration

These are the cipher suites that offer most security and compatibility and should be offered by Apache for e-banking applications. No SSLv2 and SSlv3 support should be provided at all.

Cipher suite name	*Protocol*	*KeyX*	*Auth*	*Enc*	*bit*	*Hash*	*Comp.*
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA	TLS 1.0	ECDHE	ECDSA	AES	256	SHA	
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA	TLS 1.0	ECDHE	ECDSA	AES	128	SHA	
TLS_DHE_RSA_WITH_AES_256_CBC_SHA	TLS 1.0	DHE	RSA	AES	256	SHA	
TLS_DHE_RSA_WITH_AES_128_CBC_SHA	TLS 1.0	DHE	RSA	AES	128	SHA	
TLS_RSA_WITH_RC4_128_SHA	TLS 1.0	RSA	RSA	RC4	128	SHA	
TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA	TLS 1.0	DHE	DSS	3DES	168	SHA	

Firefox3 Opera
Windows XP/2000/2003 (IE7, Safari) Windows 7/Vista/2008 (IE8, Safari)
Safari (MacOSx)

IIS6 recommended SSL configuration[29] [30]

These are the cipher suites that offer most security and compatibility and should be offered by Apache for e-banking applications. No SSLv2 and SSlv3 support should be provided at all.

Cipher suite name	*Protocol*	*KeyX*	*Auth*	*Enc*	*bit*	*Hash*	*Comp.*
TLS_DHE_RSA_WITH_AES_256_CBC_SHA*	TLS 1.0	DHE	RSA	AES	256	SHA	
TLS_DHE_RSA_WITH_AES_128_CBC_SHA*	TLS 1.0	DHE	RSA	AES	128	SHA	
TLS_RSA_WITH_RC4_128_SHA	TLS 1.0	RSA	RSA	RC4	128	SHA	
TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA	TLS 1.0	DHE	DSS	3DES	168	SHA	

* IIS6 will support AES only after the installation of a Hotfix (which is recommended)
Firefox3 Opera
Windows XP/2000/2003 (IE7, Safari) Windows 7/Vista/2008 (IE8, Safari)
Safari (MacOSx)

[29] http://support.microsoft.com/?scid=kb;en-us;245030&x=14&y=11
[30] http://www.gorlani.com/publicprj/CipherControl/

Summary

Minimum SSL configuration for E-banking

- Use a private key that is **at least 2048** bits long
  (See section "Minimal symmetric Key length")
- **Do not** offer ciphers below 128 bit
  (See section "Minimal asymmetric Key length")
- **Do not** support SSLv2
  (see section "SSLv2 Technical details")
- **Do not** offer Anonymous Diffie Hellman support (ADH)
- **Do not** reuse keys across certificates and generate new keys for every certificate you request
- **Do offer TLS 1.0 and/or better support**

Recommended SSL configuration for E-banking

- Support Elliptic key cryptography as preferred cipher
- Offer AES as encryption algorithm
- Use a minimum encryption key length 128-bit
- Use key exchanged that that offer perfect forward secrecy (DHE)
- The RSA key size needs to be **at least 2048** bits strong
- Drop support for SSLv2 and SSLv3 (See Browser compatibility chart)
- Restrict protocol support TLS 1.0 or better support
  (See Browser compatibility chart)
- Use Client certificates as an additional layer to authenticate clients

Sources

1. http://www.ssllabs.com
2. https://www.mikestoolbox.net/
3. http://extendedsubset.com/

Thanks

We would like to thank Ivan Ristic (SSL Labs) for the support.

Disclaimer

The Information is believed to be accurate by the time of writing.

Copyright

Appendix

Enumerate Crypto api

```
//////////////////////////////////////////////////////////////////////
// EnumProviders.cpp
//    Enumerate the cryptographic providers installed on the
//    computer. This sample enumerates the Cryptography API
//    (CryptoAPI) and Cryptography API: Next Generation (CNG)
//    providers.

// Specify that the minimum target operating system is Vista.
#ifndef NTDDI_VERSION
#define NTDDI_VERSION NTDDI_VISTA
#endif

#include <certenroll.h>
#include <windows.h>
#include <stdio.h>
#include <conio.h>
#include <tchar.h>
#include <atlbase.h>

// Forward declaration.
HRESULT enumProviders(void);

int _tmain(int argc, _TCHAR* argv[])
{
   HRESULT hr = S_OK;

   // Initialize COM.
   hr = CoInitializeEx(NULL, COINIT_APARTMENTTHREADED);
   if(FAILED(hr)) return hr;

   // Enumerate the CryptoAPI and CNG providers.
   hr = enumProviders();

   CoUninitialize();
   return hr;
}

HRESULT enumProviders(void)
{
   CComPtr<ICspInformations>     pCSPs;   // Provider collection
   CComPtr<ICspInformation>      pCSP;    // Provider instgance
   HRESULT           hr          = S_OK;  // Return value
   long              lCount      = 0;     // Count of providers
   CComBSTR          bstrName;            // Provider name
   VARIANT_BOOL      bLegacy;             // CryptoAPI or CNG

   // Create a collection of cryptographic providers.
   hr = CoCreateInstance(
            __uuidof(CCspInformations),
            NULL,
            CLSCTX_INPROC_SERVER,
            __uuidof(ICspInformations),
            (void **) &pCSPs);
   if(FAILED(hr)) return hr;

   // Add the providers installed on the computer.
   hr = pCSPs->AddAvailableCsps();
   if(FAILED(hr)) return hr;
```

```
    // Retrieve the number of installed providers.
    hr = pCSPs->get_Count(&lCount);
    if(FAILED(hr)) return hr;

    // Print the providers to the console. Print the
    // name and a value that specifies whether the
    // CSP is a legacy or CNG provider.
    for (long i=0; i<lCount; i++)
    {
       hr = pCSPs->get_ItemByIndex(i, &pCSP);
       if(FAILED(hr)) return hr;

       hr = pCSP->get_Name(&bstrName);
       if(FAILED(hr)) return hr;

       hr = pCSP->get_LegacyCsp(&bLegacy);
       if(FAILED(hr)) return hr;

       if(VARIANT_TRUE == bLegacy)
          wprintf_s(L"%2d. Legacy: ", i);
       else
          wprintf_s(L"%2d. CNG: ", i);

       wprintf_s(L"%s\n", static_cast<wchar_t*>(bstrName.m_str));

       pCSP=NULL;

    }

    printf_s("\n\nHit any key to continue: ");
    _getch();

    return hr;

}
```