SSL/ TLS Cipher Suite Analysis and strong Cipher Enablement

A research by

Robert Lin
Symantec, Senior Technical Engineer (14.01.2014)
Index

Introduction .................................................................page 3
1. What is a Cipher? .......................................................page 5
2. Strength and Weakness of Ciphers.................................page 7
3. Selection of cipher suite..............................................page 13
4. Checking for weak ciphers..........................................page 16
5. Enabling Ciphers......................................................page 19
6. Summary...............................................................page 22

References.................................................................page 23
Introduction

Working with SSL and PKI for many years, the most common theme that system directors, managers, and administrators communicate is that “Security is Perception”. As long as the end client has a perception that a system is secure, that is all that matters. Over the recent years however, the world has seen the devastating effect of a breached system that is not secure. Security is not merely a perception, it cannot be superficial. It is an ongoing process that must be solid and implemented correctly.

The purpose of this research is to provide an implementation process to set up a strongly secured SSL/TLS system by viewing the available cipher suites present in a system, recognizing the strength and weakness of the different ciphers and choosing the most applicable cipher suite, within the acceptable security policies, internal and external. This is not an attempt to provide an answer to that indefinitely arguable question of performance versus security.

While cryptanalysts and security experts would prefer the strongest SSL/TLS communication available, it may not always be practical when serving global clients, or when speed of delivery and client compatibilities will be impacted. On the other hand, the security flaws and attacks reported in the recent years have raised awareness that a majority of networks are still not secure enough.

(48.9% insecure- SSL pulse: https://www.trustworthyinternet.org/ssl-pulse/)
Key Findings

- Certificate Chain
  - Sites with incomplete certificate chain: 6.1% (9,634)
- Cipher Strength
  - Sites that support weak or insecure cipher suites: 29.7% (46,951)

(30.6% still using insecure cipher suites)

It is no longer a question of if an attack might occur but when and what preventative mitigation was done. This research will try to address the mitigation from a PKI point of view, but is by no means definitive. As Bruce Schneier has said, “Security is a process not a product.” What is secure today may be broken tomorrow.

The different parts of a cipher suite will be addressed, currently known weaknesses, how to check for the available ciphers on a server and how to use them accordingly. It will be a synergy of public information from different researchers, cryptanalysts and security professionals. Some of the tools mentioned in this document have been tested by me and I will provide my test results. I have no created any of these tools, so credit goes to the various developers or organizations.
1. What is a cipher?

Today SSL is an integral part of online businesses and any secured communication. It is however not an area that many system administrators or security experts have experience with. Most admins only see the certificate side of SSL/TLS handshake but neglect to make sure the strength of the connection is strong. To do this one must look at the cipher suite available on both the server and the client application.

(i) Encryption during SSL handshake

During an SSL handshake, the client sends a ClientHello message specifying the required protocol and the list of cipher suite that it is able to handle. The server then responds with a ServerHello message, containing the protocol and the strongest cipher suites that both the client and server support, together with the server certificate. Depending on the cipher suite, a session key is created to encrypt the SSL communication. If the client is a browser, a padlock appears and the protocol is shown as https://.

Here we have two encryption processes: 1) the encryption of the server message by the server’s public key. This part is known by system administrators as they have done the work of certificate enrolment and installation on the server. The strength determined by the key size of the certificate together with its corresponding private key. 2) The encryption of the SSL communication by the one-time session key that is dependent upon the chosen cipher suite between the server and the client. If either the available cipher suite on the server or client is weak, the session key will be weak and naturally the session strength will be weak too.

(ii) Components of a cipher suite

A cipher suite consists of a set of cryptographic algorithms used for the following:

a) Protect information required to create shared keys
b) Encrypt messages exchanged between client and server
c) Generate message hashes and signatures to ensure the integrity of a message

It has the following components: protocol, key exchange algorithm, authentication algorithm, symmetric encryption algorithm and hash algorithm. They are also represented as such:

Kx: Key exchange algorithm - Used to encrypt secret message containing session key
Au: Authentication algorithm - Used to authenticate server (or client)
Enc: Symmetric encryption algorithm - Encrypt communication during SSL handshake.
Mac: Hash algorithm – Provide integrity of secret message.
According to IANA standards, cipher suites are represented in this format:

**TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA**

Broken into different parts:

- **TLS**: Protocol
- **ECDHE**: Key Exchange Algorithm
- **RSA**: Authentication Algorithm
- **AES_128_CBC**: Symmetric Encryption Algorithm
- **SHA**: Hash Algorithm

Other naming standards such as OpenSSL and GnuTLS will not be used in this research.

Below is a summary table of the different algorithms more commonly used and recognized:

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Key Exchange / Authentication algorithm</th>
<th>Symmetric encryption algorithm</th>
<th>Hash algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>TLS:</td>
<td>ECDHE</td>
<td>AES_128_CBC</td>
<td>SHA-1</td>
</tr>
<tr>
<td>SSL:</td>
<td>DHE</td>
<td>RC4</td>
<td>MD5</td>
</tr>
<tr>
<td>JioSec</td>
<td>DHE</td>
<td>AES_256_CBC</td>
<td>SHA-1</td>
</tr>
<tr>
<td>Kerbros</td>
<td>ECDH</td>
<td>AES_128_CBC</td>
<td>HMAC_MD5</td>
</tr>
<tr>
<td>Internet Key Exchange</td>
<td>ECDHE</td>
<td>AES_256_CBC</td>
<td>HMAC_SHA</td>
</tr>
<tr>
<td>Point to Point</td>
<td></td>
<td>AES_256_CBC</td>
<td>SHA-256</td>
</tr>
</tbody>
</table>

**Key Exchange / Authentication algorithms:**

- **RSA**: using RSA certificates and keys
- **DH / DHE_DSS**: Using DSA certificate and keys
- **ECDHE / ECDH**: Using ECC certificate and keys

**Symmetric Encryption algorithm:**

- **DES / 3DES**: Data Encryption Standard. First encryption published by NIST. 64 bit Block cipher. Weak.
- **RC4**: Rivest 4. Stream cipher. One of the most popular cipher. Considered vulnerable.
- **AES**: Advanced Encryption Standard, also known as Rijndael (pronounced as Rain Doll). 128 bit block cipher. Recommended cipher to use by security experts today with AES-GCM.

**Hash algorithm:**

- **MD5**: Message Digest Algorithm. Weak and vulnerable. Should not be used.
- **SHA**: Secure Hash Algorithm. Most used today. SHA1 vulnerable, recommended using SHA 256.

For a complete list of cipher suites please view:

([IANA]: http://www.iana.org/assignments/tls-parameters/tls-parameters.xml)
2. Strength and weaknesses of ciphers

With the advancement in technology, the strength and weaknesses of a cipher is currently often debated. It usually depends on client compatibility, key size, faulty random number generators, cipher vulnerabilities, and including side channel attacks. While this research will not take all aspects of the debate into account, the most common factors will be touched upon to better help choose the most appropriate cipher suite for use.

OWASP considers these factors (protocol, ciphers, and keys) as weak:

1) Weak ciphers less than 128 must not be used. No Null cipher suites (no encryption), no anonymous Diffie-Hellman (no authentication)
2) Weak protocols must be disabled. E.g. SSLv2
3) Renegotiation must be secure
4) No export level cipher suites
5) Keys must at least be 1024 bit. (However according to NIST, after 2013, 2048 bit keys must be used)
6) Keys must be signed with secure hashing algorithm. E.g. no MD5 hashes.
7) Keys generated with proper entropy. E.g. No Debian Weak Keys.


NIST SP 800 - 131A contains the following:

Certificates and Key algorithms:

RSA keys less than 2048 disallowed after 2013
DSA keys less than 2048 disallowed after 2013
EC (elliptic curve) keys less than 224 disallowed after 2013
(NIST: http://www.nist.org/nist_plugins/content/content.php?content.71)

Symmetric encryption algorithms:

3DES disallowed after 2015 (However 3DES supports only 108 or 112 bits, which is below acceptable 128 bit. It is also slow and resource intensive so it is no longer a recommended cipher)
(NIST SP 800 -131A)

RC4 and AES are currently the most discussed symmetric encryption ciphers in the industry, even though both are known to have certain weaknesses.
RC4: It is currently one of the most used ciphers in the industry. There are over 92% of surveyed websites that uses or enabled RC4.

Although it has been known for many years to have a flaw (bit bias), it was only in 2013 that it was considered that a feasible exploit exists (experiment by Dan Bernstein, "On the security of RC4 in TLS and WPA"). While some security experts recommends removal of RC4 (FIPS 140 do not include RC4 as an available cipher), generally it is considered as safe.

Adam Langley (Google Online Security Blog):

“RC4 is a very common stream cipher but is showing its 26-year age.

Biases in the RC4 keystream have been known for over a decade or more and were used to attack WEP, the original security standard for Wi-Fi. HTTPS was believed to be substantially unaffected by these results until Paterson et al compiled and extended them and demonstrated that belief to be incorrect.

The best, known attack against using RC4 with HTTPS involves causing a browser to transmit many HTTP requests -- each with the same cookie -- and exploiting known biases in RC4 to build an increasingly precise probability distribution for each byte in a cookie. However, the attack needs to see on the order of 10 billion copies of the cookie in order to make a good guess. This involves the browser sending ~7TB of data. In ideal situations, this requires nearly three months to complete.

This attack cannot be mitigated without replacing RC4.”

(SSL pulse: https://www.trustworthyinternet.org/ssl-pulse/)

Dan Bernstein:

“The attacks can only be carried out by a determined attacker who can generate sufficient sessions for
the attacks. They recover a limited amount of plaintext. In this sense, the attacks do not pose a significant
danger to ordinary users of TLS or WPA/TKIP in their current form. However, it is a truism that attacks
only get better with time, and we anticipate significant further improvements to our attacks. In addition,
because of their extremely widespread use, any attacks against TLS or WPA/TKIP require careful
evaluation.”

(http://www.isg.rhul.ac.uk/tls/)

Bruce Schneier:

“The attack is very specialized:
The attack is a multi-session attack, which means that we require a target plaintext to be repeatedly sent
in the same position in the plaintext stream in multiple TLS sessions. The attack currently only targets the
first 256 bytes of the plaintext stream in sessions. Since the first 36 bytes of plaintext are formed from an
unpredictable Finished message when SHA-1 is the selected hashing algorithm in the TLS Record
Protocol, these first 36 bytes cannot be recovered. This means that the attack can recover 220 bytes of
TLS-encrypted plaintext.

The number of sessions needed to reliably recover these plaintext bytes is around $2^{30}$, but already with
only $2^{24}$ sessions, certain bytes can be recovered reliably.

Is this a big deal? Yes and no. The attack requires the identical plaintext to be repeatedly encrypted.
Normally, this would make for an impractical attack in the real world, but http messages often have
stylized headers that are identical across a conversation -- for example, cookies. On the other hand,
those are the only bits that can be decrypted. Currently, this attack is pretty raw and unoptimized -- so it’s
likely to become faster and better.

There’s no reason to panic here. But let’s start to move away from RC4 to something like AES.”

(https://www.schneier.com/blog/archives/2013/03/new_rc4_attack.html)

RC4 were most commonly still used after the press release due to the BEAST attack (Mitigating the
BEAST attack on TLS – Ivan Ristic). However most browsers today have patched against the browser
exploit and most agree to use AES instead.

AES: Adam Langley (Google Online Security Blog) on

AES - CBC:

“AES-CBC has a couple of problems, both of which are problems with the way that TLS uses CBC
(Cipher Block Chaining) mode, and not problems with AES.

The first is called BEAST and was demonstrated by Duong and Rizzo 2011 (although the idea was
originally described by Rogaway in 1995). It exploits a flaw in the way that TLS prior to version 1.1
generated CBC initialization vectors.
The attack requires precise control over the TLS connection which is not generally possible from a vanilla browser; the demo used a Java applet to obtain this control. The version of the WebSockets protocol used at the time may have allowed the necessary degree of control, but that had already been replaced by the time that the issue was demonstrated.

However, browsers are complex and evolving pieces of software, and the necessary degree of control is certainly not a comfortable barrier to exploitation. If possible, the exploit is very practical. It requires the attacker to have access to the network near the computer but otherwise completes quickly and deterministically.

The issue is fixed either by using TLS >= 1.1, or by a trick called 1/n-1 record splitting, which has been implemented by all major browsers now. However, many older installations may still exist with Java enabled and would thus be vulnerable to this attack.

The second issue is called Lucky13. This attack uses the fact that TLS servers take a slightly different amount of time to process different types of invalid TLS records. This attack is the first one that we have discussed that requires the use of timing side-channels and is thus probabilistic.

The attack needs nearly 10,000 TLS connections per byte of plaintext decoded and the attacker needs to be situated close to the TLS server in order to reduce the amount of timing noise added by the network. Under absolutely ideal situations, an attacker could extract a short (16 byte) cookie from a victim's browser in around 10 minutes. With optimistic but plausible parameters, the attack could work in an hour.

This attack can only be fixed at the server by making the decoding of all CBC records take a constant amount of time. It's not plausible for a browser to detect whether a server has fixed this issue before using AES-CBC.

AES-GCM

“There are no known breaks of AES-GCM and it is one of the ciphers that TLS servers are urged to support. However it suffers from a couple of practical issues:

The first is that it's very challenging to implement AES-GCM in software in a way which is both fast and secure. Some CPUs implement AES-GCM directly in hardware (this is called AES-NI by Intel, the most prominent example of this) and these CPUs allow for implementations that are secure and very fast, but hardware support is far from ubiquitous.

The second nit with AES-GCM is that, as integrated in TLS, implementations are free to use a random nonce value. However, the size of this nonce (8 bytes) is too small to safely support using this mode. Implementations that do so are at risk of a catastrophic nonce reuse after sending on the order of a terabyte of data on a single connection. This issue can be resolved by using a counter for the nonce but using random nonces is the most common practice at this time.

When both parties to a TLS connection support hardware AES-GCM and use counters, this cipher is essentially optimal.”

A list of secure ciphers can be found within Wikipedia, TLS:
Hash Algorithm:

MD5

- Should not be used. It is weak and there are practical known attacks.

SHA:

- SHA-1 disallowed after 2013 for digital signature. SHA-256+ are acceptable. (NIST SP 800 - 131A). It has a known weakness with collision attacks and so is no longer considered safe. Microsoft also issued a security advisory again using SHA-1.


SHA-256

- No practical known weakness. Recommended.

Side Channel Attacks that influence the use of the ciphers:

**BEAST**: Browser Exploit Against SSL/TLS – all CBC ciphers are affected.

**Lucky 13**: A variant of Serge Vaudenay's padding oracle attack that had previously thought to have been fixed, that uses a timing side-channel attack against the message authentication code (MAC) check stage in the TLS algorithm

**CRIME**: Compression Ratio Info-leak Made Easy – TLS Data Compression attack in HTTPS and SPDY protocols. TLS compression must not be enabled to prevent attack.

**BREACH**: Browser Reconnaissance and Exfiltration via Adaptive Compression of Hypertext – same as Crime except with standard HTTP compression.

Disabling TLS and HTTP compression slows the communication and so currently it is deemed as not mitigated, as most websites are configured to have the fastest traffic. As this cannot be done without http/TLS compression almost all websites are vulnerable to both CRIME and BREACH!
## Summary of Cipher’s Pro and Con:

<table>
<thead>
<tr>
<th>Key Exchange</th>
<th>Pro</th>
<th>Con</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSA</td>
<td>Widely supported</td>
<td>Many ciphers considered weak. E.g. DES, 3DES, MD5, SHA-1, Large key size - min 2048 bit</td>
</tr>
<tr>
<td>DSA</td>
<td>Gov standard FIPS</td>
<td>Less supported. Large key size - min 2048 bit</td>
</tr>
<tr>
<td>Cipher</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3DES</td>
<td>Widely supported</td>
<td>Slow and resource intensive, weak, small keys.</td>
</tr>
<tr>
<td>RC4</td>
<td>Widely supported, not vulnerable to BEAST attack. Can do 256 bit</td>
<td>bit bias exploit, to be phased out</td>
</tr>
<tr>
<td>AES-CBC</td>
<td>Widely support. Can do 256 bit. TLS 1.0</td>
<td>Vulnerable to BEAST attack and Lucky13</td>
</tr>
<tr>
<td>AES-GCM/CCM</td>
<td>Strong. TLS 1.2. No known Weakness. Can be applied on Hardware or Software</td>
<td>Less support on TLS 1.2. Challenge to implement both fast and secure if not configured correctly.</td>
</tr>
<tr>
<td>Hash</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MD5</td>
<td>Widely supported</td>
<td>Weak, considered broken. No longer used</td>
</tr>
<tr>
<td>SHA1</td>
<td>Widely supported</td>
<td>Known collision attack weakness. Not recommended by Microsoft, disallowed by NIST after 2013.</td>
</tr>
<tr>
<td>SHA-256</td>
<td>Strong. Recommended by NIST, Microsoft and the industry.</td>
<td>Less supported, more resources needed to implement.</td>
</tr>
</tbody>
</table>
3. Selection of cipher suite

In a SSL/TLS handshake, the client announces its supported preferred cipher suite, the server then chooses the suite to use. Normally the server will honor the client preference but the server can enforce its own order of preference. With the strength and weaknesses mentioned in the previous section, it is preferable to have at least the following:

i) TLS 1.0
ii) 2048 bit key and certificate (RSA and DSA)
iii) 256 bit session secret keys.
v) AES-CBC / RC4

For enhanced security (preferred by security professionals):

i) TLS 1.2
ii) ECC – 256 bit+ (384)
iii) PFS – Perfect Forward Secrecy
iv) AES-GCM – (against BEAST)

OWASP provides the following checklist when choosing a cipher suite:

- Secure Renegotiation should be enabled.
- MD5 should not be used, due to known collision attacks.
- RC4 should not be used, due to crypto-analytical attacks.
- Server should be protected from BEAST Attack.
- Server should be protected from CRIME attack, TLS compression must be disabled.
- Server should support Forward Secrecy.


Cisco – Next Generation Encryption provides a table with recommendations for Cryptographic Algorithms: http://www.cisco.com/web/about/security/intelligence/nextgen_crypto.html
List of cipher suites to use according to certificate type:

**RSA** certificates can use: RSA, DH_RSA, DHE_RSA
**DSA** certificates can use: DHE_DSS
**ECC** certificates can use: ECDH_ECDHA, ECDH_RSA, ECDHE_ECDSA, ECDHE_RSA

OpenSSL provides a list of ciphers available for the different protocols, SSLv3, TLS1, TLS1.1, and TLS1.2:

**AES ciphersuites from RFC3268, extending TLS v1.0**

- TLS_RSA_WITH_AES_128_CBC_SHA  AES128-SHA
- TLS_RSA_WITH_AES_256_CBC_SHA  AES256-SHA
- TLS_DH_DSS_WITH_AES_128_CBC_SHA  DH-DSS-AES128-SHA
- TLS_DH_DSS_WITH_AES_256_CBC_SHA  DH-DSS-AES256-SHA
- TLS_DH_RSA_WITH_AES_128_CBC_SHA  DH-RSA-AES128-SHA
- TLS_DH_RSA_WITH_AES_256_CBC_SHA  DH-RSA-AES256-SHA
- TLS_DHE_DSS_WITH_AES_128_CBC_SHA  DHE-DSS-AES128-SHA
- TLS_DHE_DSS_WITH_AES_256_CBC_SHA  DHE-DSS-AES256-SHA
- TLS_DHE_RSA_WITH_AES_128_CBC_SHA  DHE-RSA-AES128-SHA
- TLS_DHE_RSA_WITH_AES_256_CBC_SHA  DHE-RSA-AES256-SHA

**TLS v1.2 cipher suites**

- TLS_RSA_WITH_AES_128_CBC_SHA256  AES128-SHA256
- TLS_RSA_WITH_AES_256_CBC_SHA256  AES256-SHA256
- TLS_RSA_WITH_AES_128_GCM_SHA256  AES128-GCM-SHA256
- TLS_RSA_WITH_AES_256_GCM_SHA256  AES256-GCM-SHA256
- TLS_DH_RSA_WITH_AES_128_CBC_SHA256  DH-RSA-AES128-SHA256
- TLS_DH_RSA_WITH_AES_256_CBC_SHA256  DH-RSA-AES256-SHA256
- TLS_DH_RSA_WITH_AES_128_GCM_SHA256  DH-RSA-AES128-GCM-SHA256
- TLS_DH_RSA_WITH_AES_256_GCM_SHA256  DH-RSA-AES256-GCM-SHA256
- TLS_DH_DSS_WITH_AES_128_CBC_SHA256  DH-DSS-AES128-SHA256
- TLS_DH_DSS_WITH_AES_256_CBC_SHA256  DH-DSS-AES256-SHA256
- TLS_DH_DSS_WITH_AES_128_GCM_SHA256  DH-DSS-AES128-GCM-SHA256
- TLS_DH_DSS_WITH_AES_256_GCM_SHA256  DH-DSS-AES256-GCM-SHA256
- TLS_DHE_RSA_WITH_AES_128_CBC_SHA256  DHE-RSA-AES128-SHA256
- TLS_DHE_RSA_WITH_AES_256_CBC_SHA256  DHE-RSA-AES256-SHA256
- TLS_DHE_RSA_WITH_AES_128_GCM_SHA256  DHE-RSA-AES128-GCM-SHA256
- TLS_DHE_RSA_WITH_AES_256_GCM_SHA256  DHE-RSA-AES256-GCM-SHA256
- TLS_DHE_DSS_WITH_AES_128_CBC_SHA256  DHE-DSS-AES128-SHA256
- TLS_DHE_DSS_WITH_AES_256_CBC_SHA256  DHE-DSS-AES256-SHA256
TLS_DHE_DSS_WITH_AES_128_GCM_SHA256       DHE-DSS-AES128-GCM-SHA256
TLS_DHE_DSS_WITH_AES_256_GCM_SHA384       DHE-DSS-AES256-GCM-SHA384

TLS_ECDH_RSA_WITH_AES_128_CBC_SHA256       ECDH-RSA-AES128-SHA256
TLS_ECDH_RSA_WITH_AES_256_CBC_SHA384       ECDH-RSA-AES256-SHA384
TLS_ECDH_RSA_WITH_AES_128_GCM_SHA256       ECDH-RSA-AES128-GCM-SHA256
TLS_ECDH_RSA_WITH_AES_256_GCM_SHA384       ECDH-RSA-AES256-GCM-SHA384
TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA256    ECDH-ECDSA-AES128-SHA256
TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA384    ECDH-ECDSA-AES256-SHA384
TLS_ECDH_ECDSA_WITH_AES_128_GCM_SHA256    ECDH-ECDSA-AES128-GCM-SHA256
TLS_ECDH_ECDSA_WITH_AES_256_GCM_SHA384    ECDH-ECDSA-AES256-GCM-SHA384
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256    ECDHE-RSA-AES128-SHA256
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384    ECDHE-RSA-AES256-SHA384
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256    ECDHE-RSA-AES128-GCM-SHA256
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384    ECDHE-RSA-AES256-GCM-SHA384
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256    ECDHE-ECDSA-AES128-SHA256
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384    ECDHE-ECDSA-AES256-SHA384
TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256    ECDHE-ECDSA-AES128-GCM-SHA256
TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384    ECDHE-ECDSA-AES256-GCM-SHA384

For more information please visit: http://www.openssl.org/docs/apps/ciphers.html
4. Checking for weak ciphers

There are several tools that can be used to check for the ciphers used on the server. Some tools will check for weak ciphers as well.

Online tools, for example: Qualys SSL Lab – SSL Server Test
There are also applications that can be downloaded and used as a standalone tool:

- **nmap** - identify SSL service, check certificates, script for SSL vulnerabilities
- **TestSSLServer** - Java scanner
- **SSLyze** - Python script
- **OpenSSL** - check certificate status, ciphers, TLS compression
- **SSL Scan** - pen test SSL scanner + wrapper for SSL vulnerabilities
- **SSL Audit** - Perl script that follows Qualys SSL labs rating guide

For the use of some of these tools please visit OWASP:

Examples:

**nmap:**

```
Nmap done: 1 IP address (1 host up) scanned in 6.68 seconds
ubuntu@ubuntu-$ nmap --script ssl-cert,ssl-enum-ciphers -p 443 www.ssllabs.com
Starting Nmap 6.40 ( http://nmap.org ) at 2014-01-03 04:17 PST
Nmap scan report for www.ssllabs.com (173.203.79.216)
Host is up (0.11s latency).
PORT  STATE SERVICE
443/tcp open  https
  SSL-Cert: Subject: commonName=www.ssllabs.com/countryName=GB
  Issuer: commonName=StartCom Class 1 Primary Intermediate Server CA/organizationName=StartCom Ltd./countryN
  Public Key type: rsa
  Public Key bits: 2048
  Not valid after: 2014-04-27T00:39:24+00:00
  MD5:  c5ff 128e 3dbb a89a b2d1 abd7 caf5 d1f8
  SHA-1: 0be3 c403 ae34 dd41 86ca ba2e 94c4 46a3 e9f9 e27b
  ssl-enum-ciphers:
     SSLv3:
       ciphers: TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA - strong
       TLS_DHE_RSA_WITH_AES_128_CBC_SHA - strong
       TLS_DHE_RSA_WITH_AES_256_CBC_SHA - strong
       TLS_DHE_RSA_WITH_CAMELLIA_128_CBC_SHA - strong
       TLS_DHE_RSA_WITH_CAMELLIA_256_CBC_SHA - strong
       TLS_DHE_RSA_WITH_SEED_CBC_SHA - strong
       TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA - strong
       TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA - strong
       TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA - strong
       TLS_ECDHE_RSA_WITH_RC4_128_SHA - strong
       TLS_RSA_WITH_3DES_EDE_CBC_SHA - strong
       TLS_RSA_WITH_RC4_128_SHA - strong
     TLSv1.0:
       ciphers: TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA - strong
```
TestSSLServer:

```
C:\> cd program files \(x86\)\java\jre6\bin
C:\Program Files \(x86\)\java\jre6\bin> java -jar testsslserver.jar www.ssllabs.com

Supported versions: SSLv3 TLSv1.0 TLSv1.1 TLSv1.2
Delegated compression: no

Supported cipher suites (ORDER IS NOT SIGNIFICANT):
SSLv3
RSA_WITH_RC4_128_SHA
RSA_WITH_3DES_EDE_CBC_SHA
DH_RSA_WITH_3DES_EDE_CBC_SHA
DH_RSA_WITH_AES_128_CBC_SHA
DH_RSA_WITH_AES_256_CBC_SHA
DH_RSA_WITH_CAMELLIA_128_CBC_SHA
DH_RSA_WITH_CAMELLIA_256_CBC_SHA
TLS_DHE_RSA_WITH_SED_CBC_SHA
TLS_ECDHE_RSA_WITH_RC4_128_SHA
TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
TLSv1.0: idem
TLSv1.1: idem
TLSv1.2
RSA_WITH_RC4_128_SHA
RSA_WITH_3DES_EDE_CBC_SHA
DH_RSA_WITH_3DES_EDE_CBC_SHA
DH_RSA_WITH_AES_128_CBC_SHA
DH_RSA_WITH_AES_256_CBC_SHA
DH_RSA_WITH_CAMELLIA_128_CBC_SHA
DH_RSA_WITH_CAMELLIA_256_CBC_SHA
TLS_DHE_RSA_WITH_SED_CBC_SHA
TLS_ECDHE_RSA_WITH_TSH_148_SHA
TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
TLS_ECDHE_RSA_WITH_AES_192_CBC_SHA
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA512
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384

Server certificate(s):
- 0b3e423d944d4f8c26a249e444e3879c27b: EMAILADDRESS=webmaster@ssllabs.com, CN www.ssllabs.com, C=GB, O=1D.2.5.4.13=6W5ITLuw77qwoe01

Minimal encryption strength: strong encryption (96-bit or more)
Achievable encryption strength: strong encryption (96-bit or more)
HBA1 status: vulnerable
CRIME status: protected
```

C:\Program Files \(x86\)\Java\jre6\bin

5. Enabling Ciphers

**Microsoft Internet Information Systems** – via Schannel

**NOTE:** The configuration of the cipher suites in a Windows system is done via the registry. Please note that serious problems might occur if you modify the registry incorrectly. Make sure to back up the registry and consult Microsoft before any changes are made.

For more information please visit the Microsoft Article:
How to restrict the use of certain cryptographic algorithms and protocols in Schannel.dll – **Article 245030**

Entries and changes will be made via:

[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\SecurityProviders\Schannel]
"EventLogging"=dword:00000001
[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\SecurityProviders\Schannel\Ciphers]
[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\SecurityProviders\Schannel\CipherSuites]
[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\SecurityProviders\Schannel\Hashes]
[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\SecurityProviders\Schannel\KeyExchangeAlgorithms]
[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\SecurityProviders\Schannel\Protocols]

Example:

![Registry Editor Screenshot](image)

*Note: keyspec settings specified for the private key created from MMC will affect the ciphers used. For example keyspec=2 (digital signing- without keyexchange) can force DH ciphers to be used.*
Apache

Cipher suites can be configured with the follow example:

```xml
<VirtualHost *:443>
  ...
  SSLEngine on
  SSLCertificateFile /path/to/signed_certificate
  SSLCertificateChainFile /path/to/intermediate_certificate
  SSLCertificateKeyFile  /path/to/private/key
  SSLCACertificateFile /path/to/all_ca_certs
  SSLProtocol             all -SSLv2
  SSLHonorCipherOrder     on
  SSLCipherSuite          <recommended ciphersuite from top of this page>
  SSLCompression          off
  ...
</VirtualHost>
```

Ivan Ristic recommends the following for PFS and ECC:

“To deploy Forward Secrecy, you need to have both your web server and the underlying SSL/TLS library support Elliptic Curve cryptography. For Apache, Nginx, and OpenSSL, the following minimum versions will suffice:

- OpenSSL 1.0.1c+
- Apache 2.4.x
- nginx 1.0.6+ and 1.1.0+

Apache

SSLProtocol all -SSLv2 -SSLv3
SSLCipherSuite "EECDH+ECDSA+AESGCM EECDH+aRSA+AESGCM EECDH+ECDSA+SHA384 \EECDH+ECDSA+SHA256 EECDH+aRSA+SHA256 EECDH+aRSA+SHA256 EECDH+aRSA+RC4 \EECDH EDH+aRSA RC4 !aNULL !eNULL !LOW !3DES !MD5 !EXP !PSK !SRP !DSS"

Nginx

ssl_protocols TLSv1 TLSv1.1 TLSv1.2;
ssl_prefer_server_ciphers on;
ssl_ciphers "EECDH+ECDSA+AESGCM EECDH+aRSA+AESGCM EECDH+ECDSA+SHA384 \EECDH+ECDSA+SHA256 EECDH+aRSA+SHA256 EECDH+aRSA+SHA256 EECDH+aRSA+RC4 \EECDH EDH+aRSA RC4 !aNULL !eNULL !LOW !3DES !MD5 !EXP !PSK !SRP !DSS";

Note that with "!DSS" DSA will be disabled.

(More info: https://community.qualys.com/blogs/securitylabs/2013/08/05/configuring-apache-nginx-and-openssl-for-forward-secrecy)
Tomcat / Jboss

The cipher suite below caters for 1 ECC, 1 DSA, 1 RSA certificate environment.

```xml
<Connector protocol="HTTP/1.1" SSLEnabled="true" port="443" address="${jboss.bind.address}" scheme="https" secure="true" clientAuth="false"

keystoreFile="${jboss.server.home.dir}/conf/test.keystore"
keystorePass="PaSsWoRd"
sslProtocol = "TLS"

ciphers=" TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384,
TLS_DHE_DSS_WITH_AES_128_CBC_SHA256, TLS_DHE_RSA_WITH_AES_128_CBC_SHA256"

/>```

A new native provider has been added to the Java SE 7 release that provides several ECC-based algorithms (ECDSA/ECDH). For more information please view Java™ SE 7 Security Enhancements: http://docs.oracle.com/javase/7/docs/technotes/guides/security/enhancements-7.html

Apache Tomcat (ARP)

By default there is no support for TLSv1.1 or TLSv1.2 in Tomcat 7.0.47. These two patches must be applied in order to run TLSv1.1 and tlsv1.2:
https://issues.apache.org/bugzilla/attachment.cgi?id=30150
https://issues.apache.org/bugzilla/attachment.cgi?id=30166

Once patched, Tomcat (ARP) would reference the ciphers in the connectors as standard Apache:

```xml
<Connector port="8443"
protocol="HTTP/1.1"
maxThreads="200"
scheme="https" secure="true" SSLEnabled="true"
SSLCertificateFile="/home/mudassir/pay/p.pem"
SSLCertificateKeyFile="/home/mudassir/p/p-key.pem"
SSLEnabledProtocols="TLSv1.2"
SSLCACertificateFile="/home/mudassir/p/AdminCA1.pem"
SSLCipherSuite ="EECDH+ECDSA+AESGCM ECDH+RSA+AESGCM ECDH+ECDSA+SHA384 ECDH+ECDSA+SHA256 ECDH+RSA+SHA256 ECDH+RSA+EC4 ECDH EDH+RSA
RC4 !aNULL !eNULL !LOW !3DES !MD5 !EXP !PSK !SRP !DSS"

/>```

(More information on http://wiki.apache.org/tomcat/HowTo/SSLCiphers)
6. Summary

Currently the use of correct cipher suite is in a state of flux. The widespread use of older acceptable ciphers such as MD5, 3DES, and/or SHA-1, RC4 or even AES-CBC are now considered weak and although there are strong ciphers available, many servers or client applications do not support or enabled them. Thus most organizations choose the lesser of the two evils (lower performance or lower security) depending on internal and external requirements. However, as mentioned in the beginning of this research, organizations can no longer put security on lesser priority.

The enabling of the ciphers require careful configuration at least from the server side. Most security experts agree that only TLS 1.2 should be used. TLS 1.2 ciphers allow encryption strength at 256 bit, so depending on the certificate/private key type, this can be resource intensive. Compared to using RSA or DSA certificates where the private key and certificate needs to be at least 2048 bit, security experts are recommending ECC instead. On some SSL scanning sites such as the Qualys SSL Scans, ECC and the availability of ECC ciphers is the only way to achieve the highest grade.

I hope this document provides an objective view on the cipher suites to use within the PKI arena. Unfortunately it is not possible for me to include all the knowledge gathered during this research without making this a book instead. For more information on the different sections please refer to the different sources mentioned in the “References” section.
References — in alphabetical order


Bruce Schneier, https://www.schneier.com/blog/archives/2013/03/new_rc4_attack.html


Dan Bernstein, On the security of RC4 in TLS and WPA, www.isg.rhul.ac.uk/tls


Ivan Ristic - Configuring Apache, Nginx and OpenSSL for Forward Secrecy, https://community.qualys.com/blogs/securitylabs/2013/08/05/configuring-apache-nginx-and-openssl-forward-secrecy


Microsoft-restrict protocols in Schannel.dll, http://support.microsoft.com/kb/245030


NIST SP 800-131A, http://www.nist.org/nist_plugins/content/content.php?content.71


SSL Pulse, https://www.trustworthyinternet.org/ssl-pulse/

SSL, Gone in 30 Seconds - http://breachattack.com


Tools references:

TestSSLServer, http://www.bolet.org/TestSSLServer/, a java scanner - and also windows executable - includes tests for cipher suites, CRIME and BEAST

sslyze, https://github.com/iSECPartners/sslyze, is a python script to check vulnerabilities in SSL/TLS.

SSLScan, http://sourceforge.net/projects/sslscan/, a SSL Scanner and a wrapper in order to enumerate SSL vulnerabilities.

nmap, http://nmap.org/, script to identify SSL-based services and then to check Certificate and SSL/TLS vulnerabilities.

OpenSSL, www.openssl.org

SSLAudit, https://code.google.com/p/sslaudit/, a perl script/windows executable scanner which follows Qualys SSL Labs Rating Guide.