bettercrypto·org

Applied Crypto Hardening

Wolfgang Breyha, David Durvaux, Tobias Dussa, L. Aaron Kaplan, Florian Mendel, Christian Mock, Manuel Koschuch, Adi Kriegisch, Ulrich Pöschl, Ramin Sabet, Berg San, Ralf Schlatterbeck, Thomas Schreck, Aaron Zauner, Pepi Zawodsky

(University of Vienna, CERT.be, KIT-CERT, CERT.at, IAIK, coretec.at, FH Campus Wien, VRVis, MilCERT Austria, A-Trust, Runtux.com, Friedrich-Alexander University Erlangen-Nuremberg, azet.org, maclemon.at)

January 4, 2014

Do not talk unencrypted

Acknowledgements

We would like to express our thanks to the following reviewers and people who have generously offered their time and interest (in alphabetical order):

Brown, Scott
Brulebois, Cyril
Dulaunoy, Alexandre
Gühring Philipp
Grigg, Ian
Horenbeck, Maarten
Kovacic, Daniel
Lenzhofer, Stefan
Lorünser, Thomas
Millauer, Tobias
O'Brien, Hugh
Pacher, Christoph
Palfrader, Peter
Pape, Tobias (layout)
Petukhova, Anna (Logo)
Pichler, Patrick
Roeckx, Kurt
Seidl, Eva (PDF layout)
Wagner, Sebastian ("sebix")
Zangerl, Alexander

The reviewers did review parts of the document in their area of expertise; all remaining errors in this document are the sole responsibility of the primary authors.

Abstract

> *"Unfortunately, the computer security and cryptology communities have drifted apart over the last 25 years. Security people don't always understand the available crypto tools, and crypto people don't always understand the real-world problems."*
>
> — Ross Anderson in [And08]

This guide arose out of the need for system administrators to have an updated, solid, well researched and thought-through guide for configuring SSL, PGP, SSH and other cryptographic tools in the post-Snowden age. Triggered by the NSA leaks in the summer of 2013, many system administrators and IT security officers saw the need to strengthen their encryption settings. This guide is specifically written for these system administrators.

As Schneier noted in [Sch13a], it seems that intelligence agencies and adversaries on the Internet are not breaking so much the mathematics of encryption per se, but rather use software and hardware weaknesses, subvert standardization processes, plant backdoors, rig random number generators and most of all exploit careless settings in server configurations and encryption systems to listen in on private communications. Worst of all, most communication on the internet is not encrypted at all by default (for SMTP, opportunistic TLS would be a solution).

This guide can only address one aspect of securing our information systems: getting the crypto settings right to the best of the authors' current knowledge. Other attacks, as the above mentioned, require different protection schemes which are not covered in this guide. This guide is not an introduction to cryptography. For background information on cryptography and cryptoanalysis we would like to refer the reader to the the references in appendix B and C at the end of this document.

The focus of this guide is merely to give current *best practices for configuring complex cipher suites* and related parameters in a *copy & paste-able manner*. The guide tries to stay as concise as is possible for such a complex topic as cryptography. Naturally, it can not be complete. There are many excellent guides [IS12, fSidIB13, ENI13] and best practice documents available when it comes to cryptography. However none of them focuses specifically on what an average system administrator needs for hardening his or her systems' crypto settings.

This guide tries to fill this gap.

Contents

1. **Introduction** 7
   - 1.1. Audience 7
   - 1.2. Related publications 7
   - 1.3. How to read this guide 7
   - 1.4. Disclaimer and scope 8
   - 1.5. Methods 10

2. **Practical recommendations** 11
   - 2.1. Webservers 11
     - 2.1.1. Apache 11
     - 2.1.2. lighttpd 12
     - 2.1.3. nginx 14
     - 2.1.4. MS IIS 15
   - 2.2. SSH 17
     - 2.2.1. OpenSSH 17
     - 2.2.2. Cisco ASA 18
     - 2.2.3. Cisco IOS 18
   - 2.3. Mail Servers 19
     - 2.3.1. SMTP in general 19
     - 2.3.2. Dovecot 20
     - 2.3.3. cyrus-imapd 22
     - 2.3.4. Postfix 23
     - 2.3.5. Exim (based on 4.82) 25
   - 2.4. VPNs 29
     - 2.4.1. IPSec 29
     - 2.4.2. Check Point FireWall-1 31
     - 2.4.3. OpenVPN 34
     - 2.4.4. PPTP 36
     - 2.4.5. Cisco ASA 36
   - 2.5. PGP/GPG - Pretty Good Privacy 38
   - 2.6. IPMI, ILO and other lights out management solutions 39
   - 2.7. Instant Messaging Systems 39
   - 2.8. Database Systems 41
   - 2.9. Intercepting proxy solutions and reverse proxies 44

3. **Theory** 48
   - 3.1. Overview 48
   - 3.2. Cipher suites 48
     - 3.2.1. Architectural overview 49
     - 3.2.2. Forward Secrecy 50
     - 3.2.3. Recommended cipher suites 50
     - 3.2.4. Compatibility 53
     - 3.2.5. Choosing your own cipher suites 53

3.3. Random Number Generators 55
3.3.1. When random number generators fail 56
3.3.2. Linux 56
3.3.3. Recommendations 57
3.4. Keylengths 57
3.5. A note on Elliptic Curve Cryptography 59
3.6. A note on SHA-1 59
3.7. A note on Diffie Hellman Key Exchanges 60
3.8. Public Key Infrastructures 60
3.8.1. Certificate Authorities 61
3.8.2. Hardening PKI 62

A. Tools **63**
A.1. SSL & TLS 63
A.2. Keylength 64
A.3. RNGs 64
A.4. Guides 64

B. Links **65**

C. Suggested Reading **67**

D. Cipher Suite Name Cross-Reference **68**

E. Further research **77**

1. Introduction

1.1. Audience

Sysadmins. Sysadmins. Sysadmins. They are a force-multiplier.

1.2. Related publications

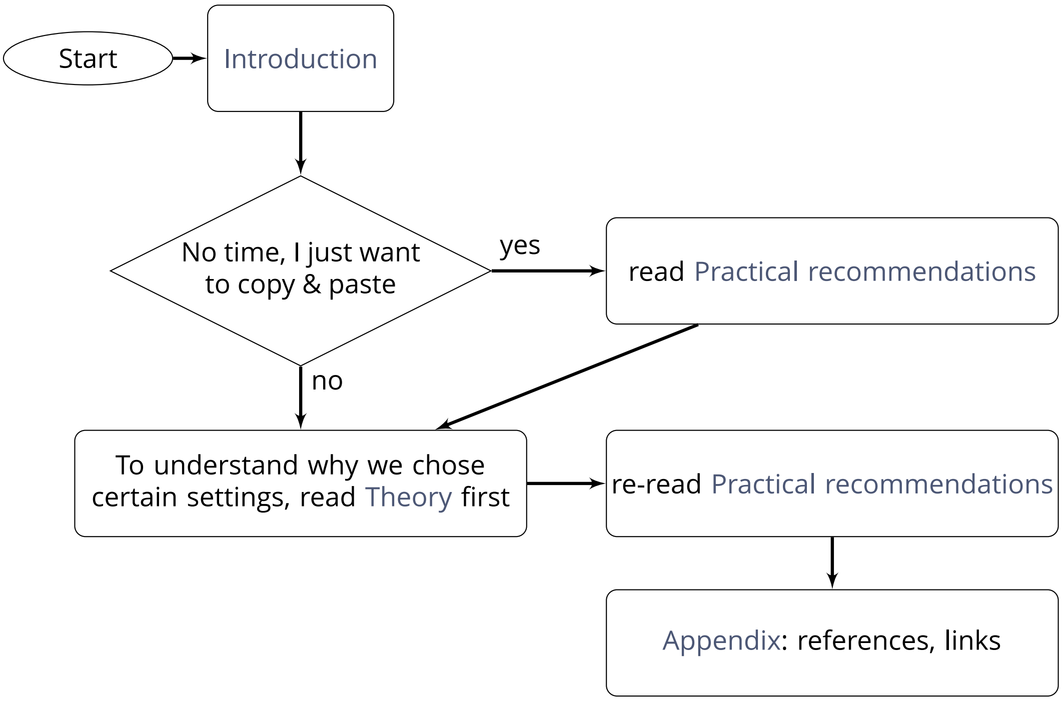
Ecrypt II [IS12], ENISA's report on Algorithms, key sizes and parameters [ENI13] and BSI's Technische Richtlinie TR-02102 [fSidIB13] are great publications which are more in depth than this guide. However, this guide has a different approach: it focuses on *copy & paste-able settings* for system administrators, effectively breaking down the complexity in the above mentioned reports to an easy to use format for the intended target audience.

1.3. How to read this guide

This guide tries to accommodate two needs: first of all, having a handy reference on how to configure the most common services's crypto settings and second of all, explaining a bit, how to chose your own cipher settings.

System administrators who want to copy & paste recommendations quickly without spending a lot of time on background reading on cryptography or cryptanalysis can do so, by simply searching for the corresponding section in chapter 2 ("Practical recommendations"). However, for the quick copy & paste approach it is important to know that this guide assumes users are happy with *cipher String B* which is the baseline and most compatible recommendation that the authors came up with. *Cipher string B* is described in 3.2.3. *Cipher String B* covers the most common use-cases (such as running an e-commerce shop, a private homepage, a mail server, . . .)

While chapter 2 is intended to serve as a copy & paste reference, chapter 3 ("Theory") explains the reasoning behind *cipher string B*. In particular, section 3.2 explains how to choose individual cipher strings. We advise the reader to actually read this section and challenge our reasoning in choosing *cipher string B* and to come up with a better or localized solution.


Start
Introduction
No time, I just want to copy & paste
yes
read Practical recommendations
no
To understand why we chose certain settings, read Theory first
re-read Practical recommendations
Appendix: references, links

1.4. Disclaimer and scope

> *"A chain is no stronger than its weakest link, and life is after all a chain"*
>
> — William James

> *"Encryption works. Properly implemented strong crypto systems are one of the few things that you can rely on. Unfortunately, endpoint security is so terrifically weak that NSA can frequently find ways around it."*
>
> — Edward Snowden, answering questions live on the Guardian's website [Gle13]

This guide specifically does not address physical security, protecting software and hardware against exploits, basic IT security housekeeping, information assurance techniques, traffic analysis attacks, issues with key-roll over and key management, securing client PCs and mobile devices (theft, loss), proper OPSec[1], social engineering attacks, anti-tempest [Wik13d] attack techniques, protecting against different side-channel attacks (timing–, cache timing–, differential fault analysis, differential power analysis or power monitoring attacks), downgrade attacks, jamming the encrypted channel or other similar attacks which are typically employed to circumvent strong encryption. The authors can not overstate the importance of these other techniques. Interested readers are advised to read about these attacks in detail since they give a lot of insight into other parts of cryptography engineering which need to be dealt with.[2]

[1] http://en.wikipedia.org/wiki/Operations_security

[2] An easy to read yet very insightful recent example is the "FLUSH+RELOAD" technique [YF13] for leaking cryptographic keys from one virtual machine to another via L3 cache timing attacks.

This guide does not talk much about the well-known insecurities of trusting a public-key infrastructure (PKI)[3]. Nor does this text fully explain how to run your own Certificate Authority (CA).

Most of this zoo of information security issues are addressed in the very comprehensive book "Security Engineering" by Ross Anderson [And08].

For some experts in cryptography this text might seem too informal. However, we strive to keep the language as non-technical as possible and fitting for our target audience: system administrators who can collectively improve the security level for all of their users.

> *"Security is a process, not a product."*
>
> — Bruce Schneier

This guide can only describe what the authors currently *believe* to be the best settings based on their personal experience and after intensive cross checking with literature and experts. For a complete list of people who reviewed this paper, see the Acknowledgements. Even though multiple specialists reviewed the guide, the authors can give *no guarantee whatsoever* that they made the right recommendations. Keep in mind that tomorrow there might be new attacks on some ciphers and many of the recommendations in this guide might turn out to be wrong. Security is a process.

We therefore recommend that system administrators keep up to date with recent topics in IT security and cryptography.

In this sense, this guide is very focused on getting the cipher strings done right even though there is much more to do in order to make a system more secure. We the authors, need this document as much as the reader needs it.

Scope

In this guide, we restricted ourselves to:

- Internet-facing services
- Commonly used services
- Devices which are used in business environments (this specifically excludes XBoxes, Playstations and similar consumer devices)
- OpenSSL

We explicitly excluded:

- Specialized systems (such as medical devices, most embedded systems, etc.)
- Wireless Access Points
- Smart-cards/chip cards

[3] Interested readers are referred to https://bugzilla.mozilla.org/show_bug.cgi?id=647959 or http://www.h-online.com/security/news/item/Honest-Achmed-asks-for-trust-1231314.html which brings the problem of trusting PKIs right to the point

1.5. Methods

> *"C.O.S.H.E.R - completely open source, headers, engineering and research*
>
> — A. Kaplan's mail signature for many years

For writing this guide, we chose to collect the most well researched facts about cryptography settings and let as many trusted specialists as possible review those settings. The review process is completely open and done on a public mailing list. The document is available (read-only) to the public Internet on the web page and the source code of this document is on a public git server, mirrored on GitHub.com and open for public scrutiny. However, write permissions to the document are only granted to vetted people. The list of reviewers can be found in the section "Acknowledgements". Every write operation to the document is logged via the "git" version control system and can thus be traced back to a specific author. We accept "git pull requests" on the github mirror[4] for this paper.

Public peer-review and multiple eyes checking of our guide is the best strategy we can imagine at the present moment [5].

We invite the gentle reader to participate in this public review process.

[4] https://github.com/BetterCrypto/Applied-Crypto-Hardening

[5] http://www.wired.com/opinion/2013/10/how-to-design-and-defend-against-the-perfect-backdoor/

2. Practical recommendations

2.1. Webservers

2.1.1. Apache

Tested with Versions

- Apache 2.4.6 linked against OpenSSL 1.0.1e, Debian jessie

Settings

Enabled modules *SSL* and *Headers* are required.

```
SSLCertificateFile server.crt
SSLCertificateKeyFile server.key
SSLProtocol All -SSLv2 -SSLv3
SSLHonorCipherOrder On
SSLCompression off
# Add six earth month HSTS header for all users...
Header add Strict-Transport-Security "max-age=15768000"
# If you want to protect all subdomains, use the following header
# ALL subdomains HAVE TO support https if you use this!
# Strict-Transport-Security: max-age=15768000 ; includeSubDomains

SSLCipherSuite 'EDH+CAMELLIA:EDH+aRSA:EECDH+aRSA+AESGCM:EECDH+aRSA+↩
    SHA384:EECDH+aRSA+SHA256:EECDH:+CAMELLIA256:+AES256:+CAMELLIA128↩
    :+AES128:+SSLv3:!aNULL:!eNULL:!LOW:!3DES:!MD5:!EXP:!PSK:!SRP:!↩
    DSS:!RC4:!SEED:!ECDSA:CAMELLIA256-SHA:AES256-SHA:CAMELLIA128-SHA↩
    :AES128-SHA'
```

Note that any cipher suite starting with EECDH can be omitted, if in doubt. (Compared to the theory section, EECDH in Apache and ECDHE in OpenSSL are synonyms [1])

[1] https://www.mail-archive.com/openssl-dev@openssl.org/msg33405.html

Additional settings

You might want to redirect everything to http**s**:// if possible. In Apache you can do this with the following setting inside of a VirtualHost environment:

```
<VirtualHost *:80>
 #...
 RewriteEngine On
      RewriteRule ^.*$ https://%{SERVER_NAME}%{REQUEST_URI} [L,R=↩
         permanent]
 #...
</VirtualHost>
```

References

https://httpd.apache.org/docs/2.4/ssl/

How to test

See appendix A

2.1.2. lighttpd

Tested with Version

- lighttpd/1.4.31-4 with OpenSSL 1.0.1e on Debian 7.0
- lighttpd/1.4.33 with OpenSSL 0.9.8o on Debian Squeeze (note that TLSv1.2 does not work in openssl 0.9.8 thus not all ciphers actually work)
- lighttpd/1.4.28-2 with OpenSSL 0.9.8o on Debian Squeeze (note that TLSv1.2 does not work in openssl 0.9.8 thus not all ciphers actually work)

Settings

TODO: FIXME: this string seems to be wrongly formatted??

```
$SERVER["socket"] == "0.0.0.0:443" {
  ssl.engine  = "enable"
  ssl.use-sslv2 = "disable"
  ssl.use-sslv3 = "disable"
  #ssl.use-compression obsolete >= 1.4.3.1
  ssl.pemfile = "/etc/lighttpd/server.pem"
```

```
    ssl.cipher-list = "EDH+CAMELLIA:EDH+aRSA:EECDH+aRSA+AESGCM:EECDH+↩
        aRSA+SHA384:EECDH+aRSA+SHA256:EECDH:+CAMELLIA256:+AES256:+↩
        CAMELLIA128:+AES128:+SSLv3:!aNULL:!eNULL:!LOW:!3DES:!MD5:!EXP↩
        :!PSK:!SRP:!DSS:!RC4:!SEED:!ECDSA:CAMELLIA256-SHA:AES256-SHA:↩
        CAMELLIA128-SHA:AES128-SHA"
    ssl.honor-cipher-order = "enable"
    setenv.add-response-header  = ( "Strict-Transport-Security" => "↩
        max-age=31536000")
  }
```

Additional settings

As for any other webserver, you might want to automatically redirect http traffic toward http**s**:// It is also recommended to set the environment variable *HTTPS*, so the applications run by the webserver can easily detect, that HTTPS is in use.

```
  $HTTP["scheme"] == "http" {
    # capture vhost name with regex conditiona -> %0 in redirect ↩
        pattern
    # must be the most inner block to the redirect rule
    $HTTP["host"] =~ ".*" {
        url.redirect = (".*" => "https://%0$0")
    }
  # Set the environment variable properly
  setenv.add-environment = (
            "HTTPS" => "on"
        )
  }
```

Additional information

The config option *honor-cipher-order* is available since 1.4.30, the supported ciphers depend on the used OpenSSL-version (at runtime). ECDH has to be available in OpenSSL at compile-time, which should be default. SSL compression should by deactivated by default at compile-time (if not, it's active).

Support for other SSL-libraries like GnuTLS will be available in the upcoming 2.x branch, which is currently under development.

References

- HTTPS redirection: http://redmine.lighttpd.net/projects/1/wiki/HowToRedirectHttpToHttps
- Lighttpd Docs SSL: http://redmine.lighttpd.net/projects/lighttpd/wiki/Docs_SSL

- Release 1.4.30 (How to mitigate BEAST attack) http://redmine.lighttpd.net/projects/lighttpd/wiki/Release-1_4_30
- SSL Compression disabled by default: http://redmine.lighttpd.net/issues/2445

How to test

See appendix A

2.1.3. nginx

Tested with Version

- 1.4.4 with OpenSSL 1.0.1e on OS X Server 10.8.5
- 1.2.1-2.2+wheezy2 with OpenSSL 1.0.1e on Debian 7.0
- 1.4.4 with OpenSSL 1.0.1e on Debian 7.0
- 1.2.1-2.2 bpo60+2 with OpenSSL 0.9.8o on Debian Squeeze (note that TLSv1.2 does not work in openssl 0.9.8 thus not all ciphers actually work)

Settings

```
ssl_prefer_server_ciphers on;
ssl_protocols TLSv1 TLSv1.1 TLSv1.2; # not possible to do exclusive
ssl_ciphers 'EDH+CAMELLIA:EDH+aRSA:EECDH+aRSA+AESGCM:EECDH+aRSA+↩
    SHA384:EECDH+aRSA+SHA256:EECDH:+CAMELLIA256:+AES256:+CAMELLIA128↩
    :+AES128:+SSLv3:!aNULL:!eNULL:!LOW:!3DES:!MD5:!EXP:!PSK:!SRP:!↩
    DSS:!RC4:!SEED:!ECDSA:CAMELLIA256-SHA:AES256-SHA:CAMELLIA128-SHA↩
    :AES128-SHA';
add_header Strict-Transport-Security max-age=2592000;
```

If you absolutely want to specify your own DH parameters, you can specify them via

```
ssl_dhparam file;
```

However, we advise you to read section 3.7 and stay with the standard IKE/IETF parameters (as long as they are > 1024 bits).

Additional settings

If you decide to trust NIST's ECC curve recommendation, you can add the following line to nginx's configuration file to select special curves:

```
ssl_ecdh_curve          secp384r1;
```

You might want to redirect everything to http**s**:// if possible. In Nginx you can do this with the following setting:

```
return 301 https://$host$request_uri;
```

References

- http://nginx.org/en/docs/http/ngx_http_ssl_module.html
- http://wiki.nginx.org/HttpSslModule

How to test

See appendix A

2.1.4. MS IIS

TODO: Daniel: add screenshots and registry keys

Tested with Version

TODO: Daniel: add tested version

Settings

When trying to avoid RC4 and CBC (BEAST-Attack) and requiring perfect forward secrecy, Microsoft Internet Information Server (IIS) supports ECDSA, but does not support RSA for key exchange (consider ECC suite B doubts[2]).

Since `ECDHE_RSA_*` is not supported, a SSL certificate based on elliptic curves needs to be used.

The configuration of cipher suites MS IIS will use, can be configured in one of the following ways:

1. Group Policy [3]
2. Registry
3. IIS Crypto [4]

[2] http://safecurves.cr.yp.to/rigid.html

[3] http://msdn.microsoft.com/en-us/library/windows/desktop/bb870930(v=vs.85).aspx

[4] https://www.nartac.com/Products/IISCrypto/

Table 2.1 shows the process of turning on one algorithm after another and the effect on the supported clients tested using https://www.ssllabs.com.

`SSL 3.0`, `SSL 2.0` and `MD5` are turned off. `TLS 1.0` and `TLS 2.0` are turned on.

Cipher Suite	Client
`TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256`	only IE 10,11, OpenSSL 1.0.1e
`TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256`	Chrome 30, Opera 17, Safari 6+
`TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA`	FF 10-24, IE 8+, Safari 5, Java 7

Table 2.1.: *Client support*

Table 2.1 shows the algorithms from strongest to weakest and why they need to be added in this order. For example insisting on SHA-2 algorithms (only first two lines) would eliminate all versions of Firefox, so the last line is needed to support this browser, but should be placed at the bottom, so capable browsers will choose the stronger SHA-2 algorithms.

`TLS_RSA_WITH_RC4_128_SHA` or equivalent should also be added if MS Terminal Server Connection is used (make sure to use this only in a trusted environment). This suite will not be used for SSL, since we do not use a RSA Key.

Clients not supported:

1. Java 6
2. WinXP
3. Bing

Additional settings

Justification for special settings (if needed)

References

TODO: add references

How to test

See appendix A

2.2. SSH

2.2.1. OpenSSH

Tested with Version

OpenSSH 6.1

Settings

```
# ...

Protocol 2
PermitEmptyPasswords no
PermitRootLogin no
StrictModes yes
HostKey /etc/ssh/ssh_host_rsa_key
Ciphers aes256-gcm@openssh.com,aes128-gcm@openssh.com,aes256-ctr,aes128-ctr
MACs umac-128-etm@openssh.com,hmac-sha2-512,hmac-sha2-256,hmac-ripemd160
KexAlgorithms curve25519-sha256@libssh.org,diffie-hellman-group-exchange-sha256,diffie-hellman-group14-sha1,diffie-hellman-group-exchange-sha1
```

Note: Older Linux systems won't support SHA2. PuTTY (Windows) does not support RIPE-MD160. Curve25519, AES-GCM and UMAC are only available upstream (OpenSSH 6.1). DSA host keys have been removed on purpose, the DSS standard does not support for DSA keys stronger than 1024bit [5] which is far below current standards (see section 3.4). Legacy systems can use this configuration and simply omit unsupported ciphers, key exchange algorithms and MACs.

References

The openssh sshd_config man page is the best reference: http://www.openssh.org/cgi-bin/man.cgi?query=sshd_config

How to test

Connect a client with verbose logging enabled to the SSH server

```
$ ssh -vvv myserver.com
```

[5] https://bugzilla.mindrot.org/show_bug.cgi?id=1647

and observe the key exchange in the output.

2.2.2. Cisco ASA

Tested with Version

9.1(3)

Settings

```
crypto key generate rsa modulus 2048
ssh version 2
ssh key-exchange group dh-group14-sha1
```

Note: When the ASA is configured for SSH, by default both SSH versions 1 and 2 are allowed. In addition to that, only a group1 DH-key-exchange is used. This should be changed to allow only SSH version 2 and to use a key-exchange with group14. The generated RSA key should be 2048 bit (the actual supported maximum). A non-cryptographic best practice is to reconfigure the lines to only allow SSH-logins.

References

http://www.cisco.com/en/US/docs/security/asa/asa91/configuration/general/admin_management.html

How to test

Connect a client with verbose logging enabled to the SSH server

```
$ ssh -vvv myserver.com
```

and observe the key exchange in the output.

2.2.3. Cisco IOS

Tested with Version

15.0, 15.1, 15.2

Settings

```
crypto key generate rsa modulus 2048 label SSH-KEYS
ip ssh rsa keypair-name SSH-KEYS
ip ssh version 2
ip ssh dh min size 2048
```

Note: Same as with the ASA, also on IOS by default both SSH versions 1 and 2 are allowed and the DH-key-exchange only use a DH-group of 768 Bit. In IOS, a dedicated Key-pair can be bound to SSH to reduce the usage of individual keys-pairs.

References

http://www.cisco.com/en/US/docs/ios/sec_user_services/configuration/guide/sec_secure_shell_v2.html

How to test

Connect a client with verbose logging enabled to the SSH server

```
$ ssh -vvv myserver.com
```

and observe the key exchange in the output.

2.3. Mail Servers

This section documents the most common mail (SMTP) and IMAPs/POPs servers. Another option to secure IMAPs/POPs servers is to place them behind an stunnel server.

2.3.1. SMTP in general

SMTP usually makes use of opportunistic TLS. This means that an MTA will accept TLS connections when asked for it during handshake but will not require it. One should always support incoming opportunistic TLS and always try TLS handshake outgoing.

Furthermore a mailserver can operate in three modes:

- As MSA (Mail Submission Agent) your mailserver receives mail from your clients MUAs (Mail User Agent).
- As receiving MTA (Mail Transmission Agent, MX)
- As sending MTA (SMTP client)

We recommend the following basic setup for all modes:

- correctly setup MX, A and PTR RRs without using CNAMEs at all.
- enable encryption (opportunistic TLS)
- do not use self signed certificates

For SMTP client mode we additionally recommend:

- the hostname used as HELO must match the PTR RR
- setup a client certificate (most server certificates are client certificates as well)
- either the common name or at least an alternate subject name of your certificate must match the PTR RR
- do not modify the cipher suite for client mode

For MSA operation we recommend:

- listen on submission port 587
- enforce SMTP AUTH even for local networks
- do not allow SMTP AUTH on unencrypted connections
- optionally use the recommended cipher suites if (and only if) all your connecting MUAs support them

We strongly recommend to allow all cipher suites for anything but MSA mode, because the alternative is plain text transmission.

2.3.2. Dovecot

Tested with Version

- Dovecot 2.2
- 2.0.19apple1 on OS X Server 10.8.5 (without ssl_prefer_server_ciphers)

Settings

```
ssl_cipher_list = 'EDH+CAMELLIA:EDH+aRSA:EECDH+aRSA+AESGCM:EECDH+↩
    aRSA+SHA384:EECDH+aRSA+SHA256:EECDH:+CAMELLIA256:+AES256:+↩
    CAMELLIA128:+AES128:+SSLv3:!aNULL:!eNULL:!LOW:!3DES:!MD5:!EXP:!↩
    PSK:!SRP:!DSS:!RC4:!SEED:!ECDSA:CAMELLIA256-SHA:AES256-SHA:↩
    CAMELLIA128-SHA:AES128-SHA'
ssl_prefer_server_ciphers = yes
```

Additional info

Dovecot 2.0, 2.1: Almost as good as dovecot 2.2. Dovecot does not ignore unknown configuration parameters. Does not support ssl_prefer_server_ciphers

Limitations

Dovecot currently does not support disabling TLS compression. Furthermore, DH parameters greater than 1024bit are not supported. The most recent version 2.2.7 of Dovecot implements configurable DH parameter length [6].

References

http://wiki2.dovecot.org/SSL

How to test

```
openssl s_client -crlf -connect SERVER.TLD:993
```

[6] http://hg.dovecot.org/dovecot-2.2/rev/43ab5abeb8f0

2.3.3. cyrus-imapd

Tested with Version

2.4.17

Settings

imapd.conf

To activate SSL/TLS configure your certificate with

```
tls_cert_file: .../cert.pem
tls_key_file: .../cert.key
```

Do not forget to add necessary intermediate certificates to the .pem file.

Limiting the ciphers provided may force (especially older) clients to connect without encryption at all! Sticking to the defaults is recommended.

If you still want to force strong encryption use

```
tls_cipher_list: EDH+CAMELLIA:EDH+aRSA:EECDH+aRSA+AESGCM:EECDH+aRSA↩
    +SHA384:EECDH+aRSA+SHA256:EECDH:+CAMELLIA256:+AES256:+↩
    CAMELLIA128:+AES128:+SSLv3:!aNULL:!eNULL:!LOW:!3DES:!MD5:!EXP:!↩
    PSK:!SRP:!DSS:!RC4:!SEED:!ECDSA:CAMELLIA256-SHA:AES256-SHA:↩
    CAMELLIA128-SHA:AES128-SHA
```

cyrus-imapd loads hardcoded 1024 bit DH parameters using get_rfc2409_prime_1024() by default. If you want to load your own DH parameters add them PEM encoded to the certificate file given in tls_cert_file. Do not forget to re-add them after updating your certificate.

To prevent unencrypted connections on the STARTTLS ports you can set

```
allowplaintext: 0
```

This way MUAs can only authenticate after STARTTLS if you only provide plaintext and SASL PLAIN login methods. Therefore providing CRAM-MD5 or DIGEST-MD5 methods is not recommended.

cyrus.conf

To support POP3/IMAP on ports 110/143 with STARTTLS add

```
imap          cmd="imapd" listen="imap" prefork=3
pop3          cmd="pop3d" listen="pop3" prefork=1
```

to the SERVICES section.

To support POP3S/IMAPS on ports 995/993 add

```
imaps         cmd="imapd -s" listen="imaps" prefork=3
pop3s         cmd="pop3d -s" listen="pop3s" prefork=1
```

Limitations

cyrus-imapd currently (2.4.17, trunk) does not support elliptic curve cryptography. Hence, ECDHE will not work even if defined in your cipher list.

Currently there is no way to prefer server ciphers or to disable compression.

There is a working patch for all three features: https://bugzilla.cyrusimap.org/show_bug.cgi?id=3823

How to test

```
openssl s_client -crlf -connect SERVER.TLD:993
```

2.3.4. Postfix

Tested with Version

Postfix 2.9.6 (Debian 7.0)

Settings

MX and SMTP client configuration

As discussed in section 2.3.1, because of opportunistic encryption we do not restrict the list of ciphers. There are still some steps needed to enable TLS, all in `main.cf`:

```
  smtpd_tls_cert_file = /etc/postfix/server.pem
  smtpd_tls_key_file = /etc/postfix/server.key
  # use 0 for Postfix >= 2.9, and 1 for earlier versions
  smtpd_tls_loglevel = 0
  # enable opportunistic TLS support in the SMTP server and client
  smtpd_tls_security_level = may
  smtp_tls_security_level = may
  # if you have authentication enabled, only offer it after STARTTLS
  smtpd_tls_auth_only = yes
  tls_ssl_options = NO_COMPRESSION
```

MSA

For the MSA `smtpd` process, we first define the ciphers that are acceptable for the "mandatory" security level, again in `main.cf`:

```
  smtpd_tls_mandatory_protocols = !SSLv2, !SSLv3
  smtpd_tls_mandatory_ciphers=high
  tls_high_cipherlist=EDH+CAMELLIA:EDH+aRSA:EECDH+aRSA+AESGCM:EECDH+↩
     aRSA+SHA384:EECDH+aRSA+SHA256:EECDH:+CAMELLIA256:+AES256:+↩
     CAMELLIA128:+AES128:+SSLv3:!aNULL:!eNULL:!LOW:!3DES:!MD5:!EXP:!↩
     PSK:!SRP:!DSS:!RC4:!SEED:!ECDSA:CAMELLIA256-SHA:AES256-SHA:↩
     CAMELLIA128-SHA:AES128-SHA
```

Then, we configure the MSA smtpd in `master.cf` with two additional options that are only used for this instance of smtpd:

```
587       inet  n       -       -       -       -       smtpd
        -o smtpd_tls_security_level=encrypt -o tls_preempt_cipherlist↩
            = yes
```

For those users who want to use ECC key exchange, it is possible to specify this via:

```
  smtpd_tls_eecdh_grade = ultra
```

Limitations

tls_ssl_options is supported from Postfix 2.11 onwards. You can leave the statement in the configuration for older versions, it will be ignored.

tls_preempt_cipherlist is supported from Postfix 2.8 onwards. Again, you can leave the statement in for older versions.

References

Refer to http://www.postfix.org/TLS_README.html for an in-depth discussion.

Additional settings

Postfix has two sets of built-in DH parameters that can be overridden with the `smtpd_tls_dh512_param_file` and `smtpd_tls_dh1024_param_file` options. The "dh512" parameters are used for export ciphers, while the "dh1024" ones are used for all other ciphers.

The "bit length" in those parameter names is just a name, so one could use stronger parameter sets; it should be possible to e.g. use the IKE Group14 parameters (see section 3.7) without much interoperability risk, but we have not tested this yet.

How to test

You can check the effect of the settings with the following command:

```
$ zegrep "TLS connection established from.*with cipher" /var/log/mail↩
   .log | awk '{printf("%s %s %s %s\n", $12, $13, $14, $15)}' | sort ↩
   | uniq -c | sort -n
      1 SSLv3 with cipher DHE-RSA-AES256-SHA
     23 TLSv1.2 with cipher DHE-RSA-AES256-GCM-SHA384
     60 TLSv1 with cipher ECDHE-RSA-AES256-SHA
    270 TLSv1.2 with cipher ECDHE-RSA-AES256-GCM-SHA384
    335 TLSv1 with cipher DHE-RSA-AES256-SHA
```

```
openssl s_client -starttls smtp -crlf -connect SERVER.TLD:25
```

2.3.5. Exim (based on 4.82)

It is highly recommended to read http://exim.org/exim-html-current/doc/html/spec_html/ch-encrypted_smtp_connections_using_tlsssl.html first.

MSA mode (submission)

In the main config section of Exim add:

```
 tls_certificate = ..../cert.pem
 tls_privatekey = ..../cert.key
```

don't forget to add intermediate certificates to the .pem file if needed.

Tell Exim to advertise STARTTLS in the EHLO answer to everyone:

```
 tls_advertise_hosts = *
```

If you want to support legacy SMTPS on port 465, and STARTTLS on smtp(25)/submission(587) ports set

```
daemon_smtp_ports = smtp : smtps : submission
tls_on_connect_ports = 465
```

It is highly recommended to limit SMTP AUTH to SSL connections only. To do so add

```
server_advertise_condition = ${if eq{$tls_cipher}{}{no}{yes}}
```

to every authenticator defined.

Add the following rules on top of your acl_smtp_mail:

```
warn    hosts            = *
        control          = submission/sender_retain
```

This switches Exim to submission mode and allows addition of missing “Message-ID” and “Date” headers.

It is not advisable to restrict the default cipher list for MSA mode if you don’t know all connecting MUAs. If you still want to define one please consult the Exim documentation or ask on the exim-users mailinglist.

The cipher used is written to the logfiles by default. You may want to add

```
log_selector = <whatever your log_selector already contains> \
 +tls_certificate_verified +tls_peerdn +tls_sni
```

to get even more TLS information logged.

server mode (incoming)

In the main config section of Exim add:

```
tls_certificate = ..../cert.pem
tls_privatekey = ..../cert.key
```

don’t forget to add intermediate certificates to the .pem file if needed.

Tell Exim to advertise STARTTLS in the EHLO answer to everyone:

```
tls_advertise_hosts = *
```

Listen on smtp(25) port only

```
daemon_smtp_ports = smtp
```

It is not advisable to restrict the default cipher list for opportunistic encryption as used by SMTP. Do not use cipher lists recommended for HTTPS! If you still want to define one please consult the Exim documentation or ask on the exim-users mailinglist.

If you want to request and verify client certificates from sending hosts set

```
tls_verify_certificates = /etc/pki/tls/certs/ca-bundle.crt
tls_try_verify_hosts = *
```

tls_try_verify_hosts only reports the result to your logfile. If you want to disconnect such clients you have to use

```
tls_verify_hosts = *
```

The cipher used is written to the logfiles by default. You may want to add

```
log_selector = <whatever your log_selector already contains> \
 +tls_certificate_verified +tls_peerdn +tls_sni
```

to get even more TLS information logged.

client mode (outgoing)

Exim uses opportunistic encryption in the SMTP transport by default.

Client mode settings have to be done in the configuration section of the smtp transport (driver = smtp).

If you want to use a client certificate (most server certificates can be used as client certificate, too) set

```
tls_certificate   = .../cert.pem
tls_privatekey    = .../cert.key
```

This is recommended for MTA-MTA traffic.

Do not limit ciphers without a very good reason. In the worst case you end up without encryption at all instead of some weak encryption. Please consult the Exim documentation if you really need to define ciphers.

OpenSSL

Exim already disables SSLv2 by default. We recommend to add

```
openssl_options = +all +no_sslv2 +no_compression +↩
   cipher_server_preference
```

to the main configuration.
Note: +all is misleading here since OpenSSL only activates the most common workarounds. But that's how SSL_OP_ALL is defined.

You do not need to set dh_parameters. Exim with OpenSSL by default uses parameter initialization with the "2048-bit MODP Group with 224-bit Prime Order Subgroup" defined in section 2.2 of RFC 5114 [LK08] (ike23). If you want to set your own DH parameters please read the TLS documentation of exim.

GnuTLS

GnuTLS is different in only some respects to OpenSSL:

- tls_require_ciphers needs a GnuTLS priority string instead of a cipher list. It is recommended to use the defaults by not defining this option. It highly depends on the version of GnuTLS used. Therefore it is not advisable to change the defaults.
- There is no option like openssl_options

Exim string expansion

Note that most of the options accept expansion strings. This way you can e.g. set cipher lists or STARTTLS advertisement conditionally. Please follow the link to the official Exim documentation to get more information.

Limitations

Exim currently (4.82) does not support elliptic curves with OpenSSL. This means that ECDHE is not used even if defined in your cipher list. There already is a working patch to provide support:
http://bugs.exim.org/show_bug.cgi?id=1397

How to test

```
openssl s_client -starttls smtp -crlf -connect SERVER.TLD:25
```

2.4. VPNs

2.4.1. IPSec

Settings

Assumptions

We assume the use of IKE (v1 or v2) and ESP for this document.

Authentication

IPSEC authentication should optimally be performed via RSA signatures, with a key size of 2048 bits or more. Configuring only the trusted CA that issued the peer certificate provides for additional protection against fake certificates.

If you need to use Pre-Shared Key authentication:

1. Choose a **random**, **long enough** PSK (see below)
2. Use a **separate** PSK for any IPSEC connection
3. Change the PSKs regularly

The size of the PSK should not be shorter than the output size of the hash algorithm used in IKE [7].

For a key composed of upper- and lowercase letters, numbers, and two additional symbols[8], table 2.2 gives the minimum lengths in characters.

IKE Hash	PSK length
SHA256	43
SHA384	64
SHA512	86

Table 2.2.: *PSK lengths*

Cryptographic Suites

[7] It is used in a HMAC, see RFC2104 [KBC97] and the discussion starting in http://www.vpnc.org/ietf-ipsec/02.ipsec/msg00268.html.

[8] 64 possible values = 6 bits

IPSEC Cryptographic Suites are pre-defined settings for all the items of a configuration; they try to provide a balanced security level and make setting up VPNs easier. [9]

When using any of those suites, make sure to enable "Perfect Forward Secrecy" for Phase 2, as this is not specified in the suites. The equivalents to the recommended ciphers suites in section 3.2.3 are shown in table 2.3.

Configuration A	Configuration B	Notes
`Suite-B-GCM-256`	`Suite-B-GCM-128` `VPN-B`	All Suite-B variants use NIST elliptic curves

Table 2.3.: *IPSEC Cryptographic Suites*

IKE or Phase 1

Alternatively to the pre-defined cipher suites, you can define your own, as described in this and the next section.

IKE or Phase 1 is the mutual authentication and key exchange phase; table 2.4 shows the parameters.

Use only "main mode", as "aggressive mode" has known security vulnerabilities [10].

	Configuration A	Configuration B
Mode	Main Mode	Main Mode
Encryption	AES-256	AES, CAMELLIA (-256 or -128)
Hash	SHA2-*	SHA2-*, SHA1
DH Group	Group 14-18	Group 14-18

Table 2.4.: *IPSEC Phase 1 parameters*

ESP or Phase 2

ESP or Phase 2 is where the actual data are protected; recommended parameters are shown in table 2.5.

References

"A Cryptographic Evaluation of IPsec", Niels Ferguson and Bruce Schneier: https://www.schneier.com/paper-ipsec.pdf

[9] RFC6379 [LS11], RFC4308 [Hof05]

[10] http://ikecrack.sourceforge.net/

	Configuration A	Configuration B
Perfect Forward Secrecy	yes	yes
Encryption	AES-GCM-16, AES-CTR, AES-CCM-16, AES-256	AES-GCM-16, AES-CTR, AES-CCM-16, AES-256, CAMELLIA-256, AES-128, CAMELLIA-128
Hash	SHA2-* (or none for AEAD)	SHA2-*, SHA1 (or none for AEAD)
DH Group	Same as Phase 1	Same as Phase 1

Table 2.5.: *IPSEC Phase 2 parameters*

2.4.2. Check Point FireWall-1

Tested with Version:

- R77 (should work with any currently supported version)

Settings:

Please see section 2.4.1 for guidance on parameter choice. In this section, we will configure a strong setup according to "Configuration A".

This is based on the concept of a "VPN Community", which has all the settings for the gateways that are included in that community. Communities can be found in the "IPSEC VPN" tab of SmartDashboard.

Either chose one of the encryption suites in the properties dialog (figure 2.1), or proceed to "Custom Encryption...", where you can set encryption and hash for Phase 1 and 2 (figure 2.2).

The Diffie-Hellman groups and Perfect Forward Secrecy Settings can be found under "Advanced Settings" / "Advanced VPN Properties" (figure 2.3).

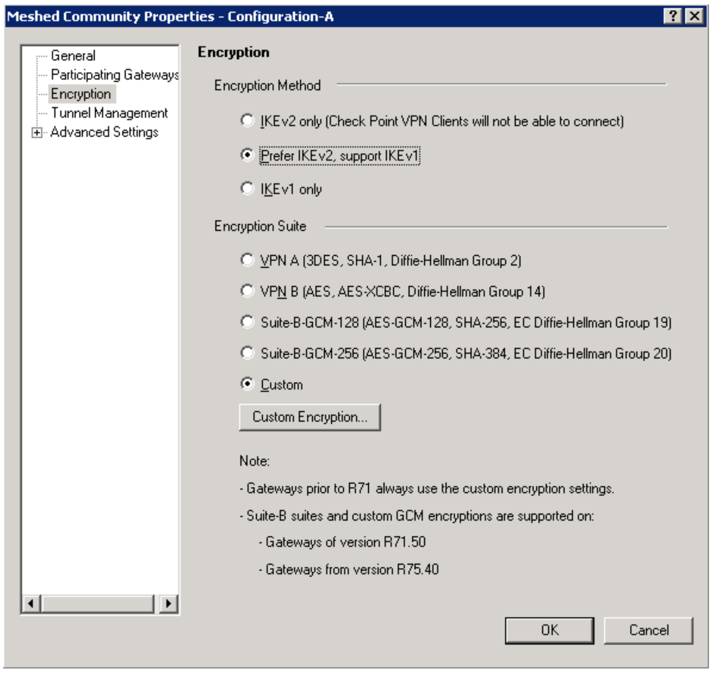
Additional settings:

For remote Dynamic IP Gateways, the settings are not taken from the community, but set in the "Global Properties" dialog under "Remote Access" / "VPN Authentication and Encryption". Via the "Edit..." button, you can configure sets of algorithms that all gateways support (figure 2.4).

Please note that these settings restrict the available algorithms for **all** gateways, and also influence the VPN client connections.

References:

- Check Point VPN R77 Administration Guide (may require a UserCenter account to access)



Encryption

Encryption Method

○ IKEv2 only (Check Point VPN Clients will not be able to connect)

● Prefer IKEv2, support IKEv1

○ IKEv1 only

Encryption Suite

○ VPN A (3DES, SHA-1, Diffie-Hellman Group 2)

○ VPN B (AES, AES-XCBC, Diffie-Hellman Group 14)

○ Suite-B-GCM-128 (AES-GCM-128, SHA-256, EC Diffie-Hellman Group 19)

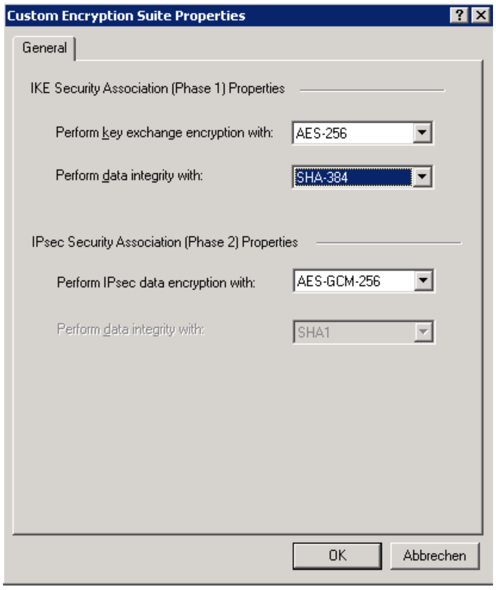
○ Suite-B-GCM-256 (AES-GCM-256, SHA-384, EC Diffie-Hellman Group 20)

● Custom

Note:

- Gateways prior to R71 always use the custom encryption settings.
- Suite-B suites and custom GCM encryptions are supported on:
  - Gateways of version R71.50
  - Gateways from version R75.40

Figure 2.1.: *VPN Community encryption properties*



IKE Security Association (Phase 1) Properties

Perform key exchange encryption with: AES-256

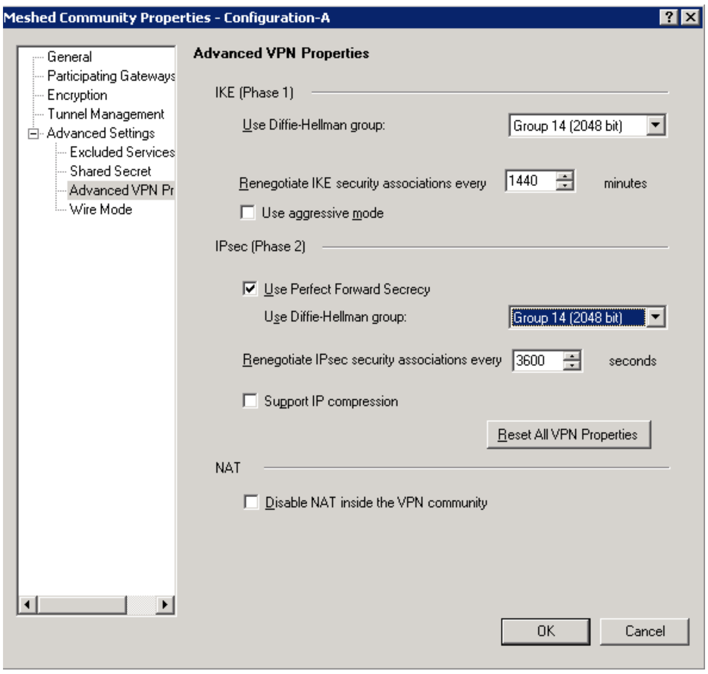
Perform data integrity with: SHA-384

IPsec Security Association (Phase 2) Properties

Perform IPsec data encryption with: AES-GCM-256

Perform data integrity with: SHA1

Figure 2.2.: *Custom Encryption Suite Properties*



Advanced VPN Properties

IKE (Phase 1)

Use Diffie-Hellman group: Group 14 (2048 bit)

Renegotiate IKE security associations every 1440 minutes

☐ Use aggressive mode

IPsec (Phase 2)

☑ Use Perfect Forward Secrecy

Use Diffie-Hellman group: Group 14 (2048 bit)

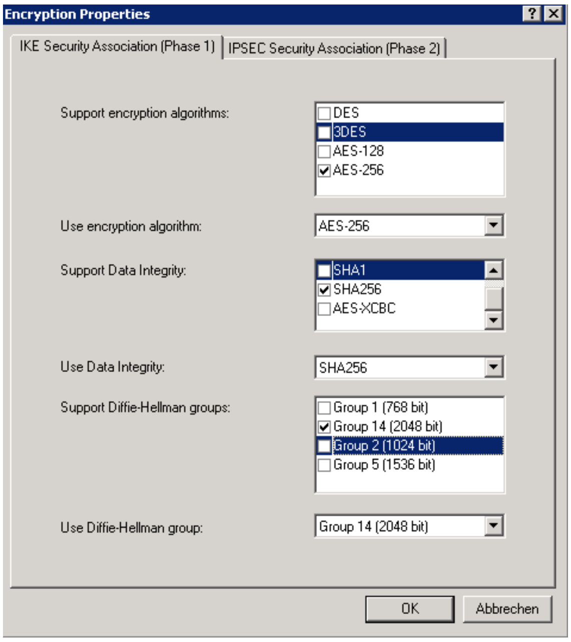
Renegotiate IPsec security associations every 3600 seconds

☐ Support IP compression

NAT

☐ Disable NAT inside the VPN community

Figure 2.3.: *Advanced VPN Properties*


Encryption Properties
IKE Security Association (Phase 1)
IPSEC Security Association (Phase 2)
OK
Abbrechen

Support encryption algorithms:
☐ DES
☐ 3DES
☐ AES-128
☑ AES-256

Use encryption algorithm: AES-256

Support Data Integrity:
☐ SHA1
☑ SHA256
☐ AES-XCBC

Use Data Integrity: SHA256

Support Diffie-Hellman groups:
☐ Group 1 (768 bit)
☑ Group 14 (2048 bit)
☐ Group 2 (1024 bit)
☐ Group 5 (1536 bit)

Use Diffie-Hellman group: Group 14 (2048 bit)

Figure 2.4.: *Remote Access Encryption Properties*

2.4.3. OpenVPN

Tested with Version:

- OpenVPN 2.3.2 from Debian “wheezy-backports” linked against openssl (libssl.so.1.0.0)
- OpenVPN 2.2.1 from Debian 7.0 linked against openssl (libssl.so.1.0.0)
- OpenVPN 2.3.2 for Windows

Settings:

General

We describe a configuration with certificate-based authentication; see below for details on the easyrsa tool to help you with that.

OpenVPN uses TLS only for authentication and key exchange. The bulk traffic is then encrypted and authenticated with the OpenVPN protocol using those keys.

Note that while the tls-cipher option takes a list of ciphers that is then negotiated as usual with TLS, the cipher and auth options both take a single argument that must match on client and server.

Server Configuration

```
tls-cipher DHE-RSA-AES256-GCM-SHA384:DHE-RSA-AES256-SHA256:DHE-↩
   RSA-AES128-GCM-SHA256:DHE-RSA-AES128-SHA256:DHE-RSA-↩
   CAMELLIA256-SHA:DHE-RSA-AES256-SHA:DHE-RSA-CAMELLIA128-SHA:↩
   DHE-RSA-AES128-SHA:CAMELLIA256-SHA:AES256-SHA:CAMELLIA128-SHA↩
   :AES128-SHA
cipher AES-256-CBC
auth SHA384
```

Client Configuration

Client and server have to use compatible configurations, otherwise they can’t communicate. The cipher and auth directives have to be identical.

```
tls-cipher DHE-RSA-AES256-GCM-SHA384:DHE-RSA-AES256-SHA256:DHE-↩
   RSA-AES128-GCM-SHA256:DHE-RSA-AES128-SHA256:DHE-RSA-↩
   CAMELLIA256-SHA:DHE-RSA-AES256-SHA:DHE-RSA-CAMELLIA128-SHA:↩
   DHE-RSA-AES128-SHA:CAMELLIA256-SHA:AES256-SHA:CAMELLIA128-SHA↩
   :AES128-SHA
cipher AES-256-CBC
auth SHA384
```

```
# http://openvpn.net/index.php/open-source/documentation/howto.↩
   html#mitm
remote-cert-tls server

tls-remote server.example.com
```

Justification for special settings (if needed):

OpenVPN 2.3.1 changed the values that the `tls-cipher` option expects from OpenSSL to IANA cipher names. That means from that version on you will get "Deprecated TLS cipher name" warnings for the configurations above. You cannot use the selection strings from section 3.2.3 directly from 2.3.1 on, which is why we give an explicit cipher list here.

In addition, there is a 256 character limit on configuration file line lengths; that limits the size of cipher suites, so we dropped all ECDHE suites.

The configuration shown above is compatible with all tested versions.

References:

http://openvpn.net/index.php/open-source/documentation/security-overview.html

Additional settings:

Key renegotiation interval

The default for renegotiation of encryption keys is one hour (`reneg-sec 3600`). If you transfer huge amounts of data over your tunnel, you might consider configuring a shorter interval, or switch to a byte- or packet-based interval (`reneg-bytes` or `reneg-pkts`).

Fixing "easy-rsa"

When installing an OpenVPN server instance, you are probably using *easy-rsa* to generate keys and certificates. The file `vars` in the easyrsa installation directory has a number of settings that should be changed to secure values:

```
export KEY_SIZE=4096
export KEY_EXPIRE=365
export CA_EXPIRE=1826
```

This will enhance the security of the key generation by using RSA keys with a length of 4096 bits, and set a lifetime of one year for the server/client certificates and five years for the CA certificate. **NOTE: 4096 bits is only an example of how to do this with easy-rsa.** See also section 3.4 for a discussion on keylengths.

In addition, edit the `pkitool` script and replace all occurences of `sha1` with `sha256`, to sign the certificates with SHA256.

Limitations:

Note that the ciphersuites shown by `openvpn --show-tls` are *known*, but not necessarily *supported* [11].

Which cipher suite is actually used can be seen in the logs:

```
Control Channel: TLSv1, cipher TLSv1/SSLv3 DHE-RSA-CAMELLIA256-SHA, 2048 bit RSA
```

2.4.4. PPTP

PPTP is considered insecure, Microsoft recommends to "use a more secure VPN tunnel"[12].

There is a cloud service that cracks the underlying MS-CHAPv2 authentication protocol for the price of USD 200[13], and given the resulting MD4 hash, all PPTP traffic for a user can be decrypted.

2.4.5. Cisco ASA

The following settings reflect our recommendations as best as possible on the Cisco ASA platform. These are - of course - just settings regarding SSL/TLS (i.e. Cisco AnyConnect) and IPSec. For further security settings regarding this platform the appropriate Cisco guides should be followed.

Tested with Version: 9.1(3) - X-series model

Settings:

```
crypto ipsec ikev2 ipsec-proposal AES-Fallback
 protocol esp encryption aes-256 aes-192 aes
 protocol esp integrity sha-512 sha-384 sha-256
crypto ipsec ikev2 ipsec-proposal AES-GCM-Fallback
 protocol esp encryption aes-gcm-256 aes-gcm-192 aes-gcm
 protocol esp integrity sha-512 sha-384 sha-256
crypto ipsec ikev2 ipsec-proposal AES128-GCM
 protocol esp encryption aes-gcm
 protocol esp integrity sha-512
crypto ipsec ikev2 ipsec-proposal AES192-GCM
 protocol esp encryption aes-gcm-192
 protocol esp integrity sha-512
crypto ipsec ikev2 ipsec-proposal AES256-GCM
 protocol esp encryption aes-gcm-256
 protocol esp integrity sha-512
crypto ipsec ikev2 ipsec-proposal AES
 protocol esp encryption aes
 protocol esp integrity sha-1 md5
crypto ipsec ikev2 ipsec-proposal AES192
```

[11] https://community.openvpn.net/openvpn/ticket/304
[12] http://technet.microsoft.com/en-us/security/advisory/2743314
[13] https://www.cloudcracker.com/blog/2012/07/29/cracking-ms-chap-v2/

```
 protocol esp encryption aes-192
 protocol esp integrity sha-1 md5
crypto ipsec ikev2 ipsec-proposal AES256
 protocol esp encryption aes-256
 protocol esp integrity sha-1 md5
crypto ipsec ikev2 sa-strength-enforcement
crypto ipsec security-association pmtu-aging infinite
crypto dynamic-map SYSTEM_DEFAULT_CRYPTO_MAP 65535 set pfs ↩
   group14
crypto dynamic-map SYSTEM_DEFAULT_CRYPTO_MAP 65535 set ikev2 ↩
   ipsec-proposal AES256-GCM AES192-GCM AES128-GCM AES-GCM-↩
   Fallback AES-Fallback
crypto map Outside-DMZ_map 65535 ipsec-isakmp dynamic ↩
   SYSTEM_DEFAULT_CRYPTO_MAP
crypto map Outside-DMZ_map interface Outside-DMZ

crypto ikev2 policy 1
 encryption aes-gcm-256
 integrity null
 group 14
 prf sha512 sha384 sha256 sha
 lifetime seconds 86400
crypto ikev2 policy 2
 encryption aes-gcm-256 aes-gcm-192 aes-gcm
 integrity null
 group 14
 prf sha512 sha384 sha256 sha
 lifetime seconds 86400
crypto ikev2 policy 3
 encryption aes-256 aes-192 aes
 integrity sha512 sha384 sha256
 group 14
 prf sha512 sha384 sha256 sha
 lifetime seconds 86400
crypto ikev2 policy 4
 encryption aes-256 aes-192 aes
 integrity sha512 sha384 sha256 sha
 group 14
 prf sha512 sha384 sha256 sha
 lifetime seconds 86400
crypto ikev2 enable Outside-DMZ client-services port 443
crypto ikev2 remote-access trustpoint ASDM_TrustPoint0

ssl server-version tlsv1-only
ssl client-version tlsv1-only
ssl encryption dhe-aes256-sha1 dhe-aes128-sha1 aes256-sha1 aes128↩
   -sha1
ssl trust-point ASDM_TrustPoint0 Outside-DMZ
```

Justification for special settings (if needed): New IPsec policies have been defined which do not make use of ciphers that may be cause for concern. Policies have a "Fallback" option to support legacy devices.

3DES has been completely disabled as such Windows XP AnyConnect Clients will no longer be able to connect.

The Cisco ASA platform does not currently support RSA Keys above 2048bits.

Legacy ASA models (e.g. 5505, 5510, 5520, 5540, 5550) do not offer the possibility to configure for SHA256/SHA384/SHA512 nor AES-GCM for IKEv2 proposals.

References: http://www.cisco.com/en/US/docs/security/asa/roadmap/asaroadmap.html
http://www.cisco.com/web/about/security/intelligence/nextgen_crypto.html

2.5. PGP/GPG - Pretty Good Privacy

The OpenPGP protocol [14] uses asymmetric encryption to protect a session key which is used to encrypt a message. Additionally, it signs messages via asymmetric encryption and hash functions. Research on SHA-1 conducted back in 2005[15] has made clear that collision attacks are a real threat to the security of the SHA-1 hash function. PGP settings should be adapted to avoid using SHA-1.

When using PGP, there are a couple of things to take care of:

- keylengths (see section 3.4)
- randomness (see section 3.3)
- preference of symmetric encryption algorithm (see section 3.2)
- preference of hash function (see section 3.2)

Properly dealing with key material, passphrases and the web-of-trust is outside of the scope of this document. The GnuPG website[16] has a good tutorial on PGP.

Hashing

Avoid SHA-1 in GnuPG. Edit $HOME/.gnupg/gpg.conf:

```
# according to:  https://www.debian-administration.org/users/dkg/↩
   weblog/48
personal-digest-preferences SHA256
cert-digest-algo SHA256
default-preference-list SHA512 SHA384 SHA256 SHA224 AES256 AES192 AES↩
    CAST5 ZLIB BZIP2 ZIP Uncompressed
```

Before you generate a new PGP key, make sure there is enough entropy available (see subsection 3.3.2).

[14] http://tools.ietf.org/search/rfc4880
[15] https://www.schneier.com/blog/archives/2005/02/sha1_broken.html
[16] http://www.gnupg.org/

2.6. IPMI, ILO and other lights out management solutions

We *strongly* recommend that any remote management system for servers such as ILO, iDRAC, IPMI based solutions and similar systems *never* be connected to the public internet. Consider creating an unrouted management VLAN and access that only via VPN.

2.7. Instant Messaging Systems

XMPP / Jabber

General server configuration recommendations

For servers, we mostly recommend to apply what's proposed by the *Peter's manifesto*[17].

In short:

- require the use of TLS for both client-to-server and server-to-server connections
- prefer or require TLS cipher suites that enable forward secrecy
- deploy certificates issued by well-known and widely-deployed certification authorities (CAs)

The last point being out-of-scope for this section, we will only cover the first two points.

ejabberd

Tested with Version: Debian 7.0, version 2.1.10-4+deb7u1

Settings:

ejabberd is one of the popular Jabber servers. In order to be compliant with the manifesto, you should adapt your configuration[18]:

```
{listen,
 [
  {5222, ejabberd_c2s, [
                        {access, c2s},
                        {shaper, c2s_shaper},
                        {max_stanza_size, 65536},
                        starttls,
                        starttls_required,
                        {certfile, "/etc/ejabberd/ejabberd.pem"}
                       ]},
  {5269, ejabberd_s2s_in, [
```

[17] https://github.com/stpeter/manifesto

[18] http://www.process-one.net/docs/ejabberd/guide_en.html

```
                           {shaper, s2s_shaper},
                           {max_stanza_size, 131072}
                          ]},

  %%% Other input ports
]}.
{s2s_use_starttls, required_trusted}.
{s2s_certfile, "/etc/ejabberd/ejabberd.pem"}.
```

Additional settings: Older Versions of ejabberd (< 2.0.0) need to be patched[19] to be able to parse all of the certificates in the CA chain.

Newer versions of ejabberd now support specifying the cipher string in the config file. See the commit message: https://github.com/processone/ejabberd/commit/1dd94ac0d06822daa8c394ea2da20d91c However, this change did not yet make it into the stable release at the time of this writing.

References:

How to test: https://xmpp.net is a practical website to test Jabber server configurations.

Chat privacy - Off-the-Record Messaging (OTR)

The OTR protocol works on top of the Jabber protocol[20]. It adds to popular chat clients (Adium, Pidgin...) the following properties for encrypted chats:

- Authentication
- Integrity
- Confidentiality
- Forward secrecy

It basically uses Diffie-Hellman, AES and SHA1. Communicating over an insecure instant messaging network, OTR can be used for end to end encryption.

There are no specific configurations required but the protocol itself is worth to be mentioned.

IRC

There are numerous implementations of IRC servers. In this section, we choose *Charybdis* which serves as basis for *ircd-seven*[21], developed and used by freenode. Freenode is actually the biggest IRC network[22]. *Charybdis* is part of the *Debian* & *Ubuntu* distributions.

[19] http://hyperstruct.net/2007/06/20/installing-the-startcom-ssl-certificate-in-ejabberd/
[20] https://otr.cypherpunks.ca/Protocol-v3-4.0.0.html
[21] https://dev.freenode.net/redmine/projects/ircd-seven
[22] http://irc.netsplit.de/networks/top10.php

```
/* Extensions */
# Some modules
#loadmodule "extensions/chm_sslonly_compat.so";
loadmodule "extensions/extb_ssl.so";
# Some other modules

serverinfo {
  /* Standard piece of information */

  ssl_private_key = "etc/test.key";
  ssl_cert = "etc/test.cert";
  ssl_dh_params = "etc/dh.pem";
  # set ssld_count as number of cores - 1
  ssld_count = 1;
};

listen {
  /* Standard ports */
  sslport = 6697;

  /* IPv6 configuration */
};
```

SILC

SILC[23] is instant messaging protocol publicly released in 2000. SILC is a per-default secure chat protocol thanks to a generalized usage of symmetric encryption. Keys are generated by the server meaning that if compromised, communication could be compromised.

The protocol is not really popular anymore.

2.8. Database Systems

Oracle

Tested with Version: not tested

References: (German) http://www.telekom.com/static/-/155996/7/technische-sicherheitsanforderungen-si

Please read the following pages about SSL and ciphersuites:
p. 129 -Req 396 and Req 397

[23] http://www.silcnet.org/ and https://en.wikipedia.org/wiki/SILC_(protocol)

MySQL

Tested with Version: Debian 7.0 and MySQL 5.5

Settings:

my.cnf

```
[mysqld]
ssl
ssl-ca=/etc/mysql/ssl/ca-cert.pem
ssl-cert=/etc/mysql/ssl/server-cert.pem
ssl-key=/etc/mysql/ssl/server-key.pem
ssl-cipher=EECDH+aRSA+AESGCM:EECDH+aRSA+SHA384:EECDH+aRSA+SHA256:↩
    EDH+CAMELLIA256:EECDH:EDH+aRSA:+SSLv3:!aNULL:!eNULL:!LOW:!3↩
    DES:!MD5:!EXP:!PSK:!SRP:!DSS:!RC4:!SEED:!AES128:!CAMELLIA128↩
    :!ECDSA:AES256-SHA
```

Additional settings:

Justification for special settings (if needed):

References: https://dev.mysql.com/doc/refman/5.5/en/ssl-connections.html

How to test: After restarting the server run the following query to see if the ssl settings are correct:

```
show variables like '%ssl%';
```

DB2

Tested with Version: not tested

References: http://pic.dhe.ibm.com/infocenter/db2luw/v9r7/index.jsp?topic=%2Fcom.ibm.db2.luw.admin.sec.doc%2Fdoc%2Fc0053544.html

ssl_cipherspecs

In the link above the whole SSL-configuration is described in-depth. The following command shows only how to set the recommended ciphersuites.

```
# recommended and supported ciphersuites

db2 update dbm cfg using SSL_CIPHERSPECS
TLS_RSA_WITH_AES_256_CBC_SHA256,
TLS_RSA_WITH_AES_128_GCM_SHA256,
TLS_RSA_WITH_AES_128_CBC_SHA256,
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256,
TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256,
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256,
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256,
TLS_RSA_WITH_AES_256_GCM_SHA384,
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384,
TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384,
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384,
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384,
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA,
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA,
TLS_RSA_WITH_AES_256_CBC_SHA,
TLS_RSA_WITH_AES_128_CBC_SHA,
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA,
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA
```

PostgreSQL

Tested with Version: Debian 7.0 and PostgreSQL 9.1

References: It's recommended to read

http://www.postgresql.org/docs/current/static/runtime-config-connection.html#RUNTIME-CONFIG-CONNECTION-SECUR
http://www.postgresql.org/docs/current/static/ssl-tcp.html http://www.postgresql.org/docs/current/static/auth-pg-hba-conf.html

Settings:

To start in SSL mode the server.crt and server.key must exist in the server's data directory $PGDATA.

Starting with version 9.2, you have the possibility to set the path.

```
ssl_key_file = '/your/path/server.key'
ssl_cert_file = '/your/path/server.crt'
ssl_ca_file = '/your/path/root.crt'
```

postgresql.conf

```
#>=8.3
ssl = on
ssl_ciphers = 'EECDH+aRSA+AESGCM:EECDH+aRSA+SHA384:EECDH+aRSA+↩
   SHA256:EDH+CAMELLIA256:EECDH:EDH+aRSA:+SSLv3:!aNULL:!eNULL:!↩
   LOW:!3DES:!MD5:!EXP:!PSK:!SRP:!DSS:!RC4:!SEED:!AES128:!↩
   CAMELLIA128:!ECDSA:AES256-SHA'
```

How to test: To test your ssl settings, run psql with the sslmode parameter:

```
psql "sslmode=require host=postgres-server dbname=database" your-↩
   username
```

2.9. Intercepting proxy solutions and reverse proxies

Within enterprise networks and corporations with increased levels of paranoia or at least some defined security requirements it is common **not** to allow direct connections to the public internet.

For this reason proxy solutions are deployed on corporate networks to intercept and scan the traffic for potential threats within sessions.

For encrypted traffic there are four options:

- Block the connection because it cannot be scanned for threats.
- Bypass the threat-mitigation and pass the encrypted session to the client, which results in a situation where malicious content is transferred directly to the client without visibility to the security system.
- Intercept (i.e. terminate) the session at the proxy, scan there and re-encrypt the session towards the client (effectively MITM).
- Deploy special Certificate Authorities to enable Deep Packet Inspection on the wire.

While the latest solution might be the most "up to date", it arises a new front in the context of this paper, because the most secure part of a client's connection could only be within the corporate network, if the proxy-server handles the connection to the destination server in an insecure manner.

Conclusion: Don't forget to check your proxy solutions SSL-capabilities. Also do so for your reverse proxies!

squid

As of squid-3.2.7 (01 Feb 2013) there is support for the OpenSSL NO_Compression option within squid config (CRIME attack) and if you combine that in the config file, with an enforcement of the server cipher preferences (BEAST Attack) you are safe.

squid.conf

TODO: UNTESTED!

```
options=NO_SSLv2,NO_TLSv1,NO_Compression,CIPHER_SERVER_PREFERENCE
cipher=EDH+CAMELLIA:EDH+aRSA:EECDH+aRSA+AESGCM:EECDH+aRSA+SHA384:↩
   EECDH+aRSA+SHA256:EECDH:+CAMELLIA256:+AES256:+CAMELLIA128:+AES128↩
   :+SSLv3:!aNULL:!eNULL:!LOW:!3DES:!MD5:!EXP:!PSK:!SRP:!DSS:!RC4:!↩
   SEED:!ECDSA:CAMELLIA256-SHA:AES256-SHA:CAMELLIA128-SHA:AES128-SHA
```

squid.conf

TODO: UNTESTED!

```
    NO_SSLv2    Disallow the use of SSLv2
    NO_SSLv3    Disallow the use of SSLv3
    NO_TLSv1    Disallow the use of TLSv1.0
    NO_TLSv1_1  Disallow the use of TLSv1.1
    NO_TLSv1_2  Disallow the use of TLSv1.2
    SINGLE_DH_USE
        Always create a new key when using temporary/ephemeral
        DH key exchanges
```

TODO: Patch here? Definitely working for 3.2.6! For squid Versions before 3.2.7 use this patch against a vanilla source-tree:

```
--- support.cc.ini      2013-01-09 02:41:51.000000000 +0100
+++ support.cc  2013-01-21 16:13:32.549383848 +0100
@@ -400,6 +400,11 @@
         "NO_TLSv1_2", SSL_OP_NO_TLSv1_2
     },
 #endif
+#ifdef SSL_OP_NO_COMPRESSION
+    {
+        "NO_Compression", SSL_OP_NO_COMPRESSION
+    },
+#endif
     {
         "", 0
     },
```

Bluecoat

Tested with Version: SGOS 6.5.x

> BlueCoat Proxy SG Appliances can be used as forward and reverse proxies. The reverse proxy feature is rather under-developed, and while it is possible and supported, there only seems to be limited use of this feature "in the wild" - nonetheless there are a few cipher suites to choose from, when enabling SSL features.

Only allow TLS 1.0,1.1 and 1.2 protocols:

```
$conf t
$(config)ssl
$(config ssl)edit ssl-device-profile default
$(config device-profile default)protocol tlsv1 tlsv1.1 tlsv1.2
  ok
```

Select your accepted cipher-suites:

```
$conf t
Enter configuration commands, one per line.  End with CTRL-Z.
$(config)proxy-services
$(config proxy-services)edit ReverseProxyHighCipher
$(config ReverseProxyHighCipher)attribute cipher-suite
Cipher#  Use         Description        Strength
-------  ---  -----------------------  ---------
      1  yes            AES128-SHA256       High
      2  yes            AES256-SHA256       High
      3  yes               AES128-SHA     Medium
      4  yes               AES256-SHA       High
      5  yes       DHE-RSA-AES128-SHA       High
      6  yes       DHE-RSA-AES256-SHA       High
                [...]
     13  yes          EXP-RC2-CBC-MD5     Export

Select cipher numbers to use, separated by commas: 2,5,6
  ok
```

The same protocols are available for forward proxy settings and should be adjusted accordingly: In your local policy file add the following section:

```
<ssl>
    DENY server.connection.negotiated_ssl_version=(SSLV2, SSLV3)
```

Disabling protocols and ciphers in a forward proxy environment could lead to unexpected results on certain (misconfigured?) webservers (i.e. ones accepting only SSLv2/3 protocol connections)

Pound

Pound 2.6

```
# HTTP Listener, redirects to HTTPS
ListenHTTP
    Address 10.10.0.10
    Port    80
    Service
        Redirect "https://some.site.tld
    End
End
## HTTPS Listener
ListenHTTPS
    Address      10.10.0.10
    Port         443
    AddHeader    "Front-End-Https: on"
    Cert         "/path/to/your/cert.pem"
    ## See 'man ciphers'.
    Ciphers      "TLSv1.2:TLSv1.1:!SSLv3:!SSLv2:EDH+CAMELLIA:EDH+aRSA↩
       :EECDH+aRSA+AESGCM:EECDH+aRSA+SHA384:EECDH+aRSA+SHA256:EECDH:+↩
       CAMELLIA256:+AES256:+CAMELLIA128:+AES128:+SSLv3:!aNULL:!eNULL↩
       :!LOW:!3DES:!MD5:!EXP:!PSK:!SRP:!DSS:!RC4:!SEED:!ECDSA:↩
       CAMELLIA256-SHA:AES256-SHA:CAMELLIA128-SHA:AES128-SHA"
    Service
        BackEnd
            Address 10.20.0.10
            Port 80
        End
    End
End
```

3. Theory

3.1. Overview

> *"The balance between freedom and security is a delicate one."*
>
> — Mark Udall, american politician

This chapter provides the necessary background information on why chapter 2 recommended *cipher string B*.

We start off by explaining the structure of cipher strings in section 3.2.1 (architecture) and define Perfect Forward Secrecy (PFS) in 3.2.2. Next we present *Cipher String A* and *Cipher String B* in section 3.2.3. This concludes the section on cipher strings. In theory, the reader should now be able to construct his or her own cipher string. However, the question why certain settings were chosen still remains. To answer this part, we need to look at recommended keylengths, problems in specific algorithms and hash functions and other cryptographic parameters. As mentioned initially in section 1.2, the ENISA [ENI13], ECRYPT 2 [IS12] and BSI [fSidIB13] reports go much more into these topics and should be consulted in addition.

We try to answer the questions by explaining issues with random number generators (section 3.3), keylengths (section 3.4), current issues in ECC (section 3.5), a note of warning on SHA-1 (section 3.6) and some comments on Diffie Hellman key exchanges (section 3.7). All of this is important in understanding why certain choices were made for *Cipher String A and B*. However, for most system administrators, the question of compatibility is one of the most pressing ones. Having the freedom to be compatible with any client (even running on outdated operating systems) of course, reduces the security of our cipher strings. We address these topics in section 3.2.4. All these sections will allow a system administrator to balance his or her needs for strong encryption with usability and compatibility.

Last but not least, we finish this chapter by talking about issues in PKIs (section 3.8), Certificate Authorities and on hardening a PKI. Note that these last few topics deserve a book on their own. Hence this guide can only mention a few current topics in this area.

3.2. Cipher suites

TODO: team: section 3.2 is currently a bit messy. Re-do it

3.2.1. Architectural overview

This section defines some terms which will be used throughout this guide.

A cipher suite is a standardized collection of key exchange algorithms, encryption algorithms (ciphers) and Message authentication codes (MAC) algorithm that provides authenticated encryption schemes. It consists of the following components:

Key exchange protocol: "An (interactive) key exchange protocol is a method whereby parties who do not share any secret information can generate a shared, secret key by communicating over a public channel. The main property guaranteed here is that an eavesdropping adversary who sees all the messages sent over the communication line does not learn anything about the resulting secret key." [KL08]

Example: `DHE`

Authentication: The client authenticates the server by its certificate. Optionally the server may authenticate the client certificate.

Example: `RSA`

Cipher: The cipher is used to encrypt the message stream. It also contains the key size and mode used by the suite.

Example: `AES256`

Message authentication code (MAC): A MAC ensures that the message has not been tampered with (integrity).

Examples: `SHA256`

Authenticated Encryption with Associated Data (AEAD): AEAD is a class of authenticated encryption block-cipher modes which take care of encryption as well as authentication (e.g. GCM, CCM mode).

Example: `AES256-GCM`

`DHE`-`RSA`-`AES256`-`SHA256`

Figure 3.1.: *Composition of a typical cipher string*

A note on nomenclature: there are two common naming schemes for cipher strings – IANA names (see appendix B) and the more well known OpenSSL names. In this document we will always use OpenSSL names unless a specific service uses IANA names.

3.2.2. Forward Secrecy

Forward Secrecy or Perfect Forward Secrecy is a property of a cipher suite that ensures confidentiality even if the server key has been compromised. Thus if traffic has been recorded it can not be decrypted even if an adversary has got hold of the server key [1] [2] [3].

3.2.3. Recommended cipher suites

In principle system administrators who want to improve their communication security have to make a difficult decision between effectively locking out some users and keeping high cipher suite security while supporting as many users as possible. The website https://www.ssllabs.com/ gives administrators and security engineers a tool to test their setup and compare compatibility with clients. The authors made use of ssllabs.com to arrive at a set of cipher suites which we will recommend throughout this document.

Configuration A: Strong ciphers, fewer clients

At the time of writing, our recommendation is to use the following set of strong cipher suites which may be useful in an environment where one does not depend on many, different clients and where compatibility is not a big issue. An example of such an environment might be machine-to-machine communication or corporate deployments where software that is to be used can be defined without restrictions.

We arrived at this set of cipher suites by selecting:

- TLS 1.2
- Perfect forward secrecy / ephemeral Diffie Hellman
- strong MACs (SHA-2) or
- GCM as Authenticated Encryption scheme

This results in the OpenSSL string:

```
'EDH+aRSA+AES256:EECDH+aRSA+AES256:!SSLv3'
```

[1] http://en.wikipedia.org/wiki/Forward_secrecy
[2] https://www.eff.org/deeplinks/2013/08/pushing-perfect-forward-secrecy-important-web-privacy-protection
[3] http://news.netcraft.com/archives/2013/06/25/ssl-intercepted-today-decrypted-tomorrow.html

ID	OpenSSL Name	Version	KeyEx	Auth	Cipher	MAC
0x009F	DHE-RSA-AES256-GCM-SHA384	TLSv1.2	DH	RSA	AESGCM(256)	AEAD
0x006B	DHE-RSA-AES256-SHA256	TLSv1.2	DH	RSA	AES(256) (CBC)	SHA256
0xC030	ECDHE-RSA-AES256-GCM-SHA384	TLSv1.2	ECDH	RSA	AESGCM(256)	AEAD
0xC028	ECDHE-RSA-AES256-SHA384	TLSv1.2	ECDH	RSA	AES(256) (CBC)	SHA384

Compatibility: Only clients which support TLS 1.2 are covered by these cipher suites (Chrome 30, Win 7 and Win 8.1 crypto stack, Opera 17, OpenSSL $\geq$ 1.0.1e, Safari 6 / iOS 6.0.1, Safari 7 / OS X 10.9).

Configuration B: Weaker ciphers but better compatibility

In this section we propose a slightly weaker set of cipher suites. For example, there are known weaknesses for the SHA-1 hash function that is included in this set. The advantage of this set of cipher suites is not only better compatibility with a broad range of clients, but also less computational workload on the provisioning hardware.

All further examples in this publication use Configuration B.

We arrived at this set of cipher suites by selecting:

- TLS 1.2, TLS 1.1, TLS 1.0
- allowing SHA-1 (see the comments on SHA-1 in section 3.6)

This results in the OpenSSL string:

```
EDH+CAMELLIA:EDH+aRSA:EECDH+aRSA+AESGCM:EECDH+aRSA+SHA384:EECDH+aRSA+↩
    SHA256:EECDH:+CAMELLIA256:+AES256:+CAMELLIA128:+AES128:+SSLv3:!↩
    aNULL:!eNULL:!LOW:!3DES:!MD5:!EXP:!PSK:!SRP:!DSS:!RC4:!SEED:!ECDSA↩
    :CAMELLIA256-SHA:AES256-SHA:CAMELLIA128-SHA:AES128-SHA
```

TODO: make a column for cipher chaining mode

ID	OpenSSL Name	Version	KeyEx	Auth	Cipher	MAC
`0x009F`	DHE-RSA-AES256-GCM-SHA384	TLSv1.2	DH	RSA	AESGCM(256)	AEAD
`0x006B`	DHE-RSA-AES256-SHA256	TLSv1.2	DH	RSA	AES(256)	SHA256
`0xC030`	ECDHE-RSA-AES256-GCM-SHA384	TLSv1.2	ECDH	RSA	AESGCM(256)	AEAD
`0xC028`	ECDHE-RSA-AES256-SHA384	TLSv1.2	ECDH	RSA	AES(256)	SHA384
`0x009E`	DHE-RSA-AES128-GCM-SHA256	TLSv1.2	DH	RSA	AESGCM(128)	AEAD
`0x0067`	DHE-RSA-AES128-SHA256	TLSv1.2	DH	RSA	AES(128)	SHA256
`0xC02F`	ECDHE-RSA-AES128-GCM-SHA256	TLSv1.2	ECDH	RSA	AESGCM(128)	AEAD
`0xC027`	ECDHE-RSA-AES128-SHA256	TLSv1.2	ECDH	RSA	AES(128)	SHA256
`0x0088`	DHE-RSA-CAMELLIA256-SHA	SSLv3	DH	RSA	Camellia(256)	SHA1
`0x0039`	DHE-RSA-AES256-SHA	SSLv3	DH	RSA	AES(256)	SHA1
`0xC014`	ECDHE-RSA-AES256-SHA	SSLv3	ECDH	RSA	AES(256)	SHA1
`0x0045`	DHE-RSA-CAMELLIA128-SHA	SSLv3	DH	RSA	Camellia(128)	SHA1
`0x0033`	DHE-RSA-AES128-SHA	SSLv3	DH	RSA	AES(128)	SHA1
`0xC013`	ECDHE-RSA-AES128-SHA	SSLv3	ECDH	RSA	AES(128)	SHA1
`0x0084`	CAMELLIA256-SHA	SSLv3	RSA	RSA	Camellia(256)	SHA1
`0x0035`	AES256-SHA	SSLv3	RSA	RSA	AES(256)	SHA1
`0x0041`	CAMELLIA128-SHA	SSLv3	RSA	RSA	Camellia(128)	SHA1
`0x002F`	AES128-SHA	SSLv3	RSA	RSA	AES(128)	SHA1

Compatibility: Note that these cipher suites will not work with Windows XP's crypto stack (e.g. IE, Outlook), We could not verify yet if installing JCE also fixes the Java 7 DH-parameter length limitation (1024 bit). TODO: do that!

Explanation: For a detailed explanation of the cipher suites chosen, please see 3.2.5. In short, finding a single perfect cipher string is practically impossible and there must be a tradeoff between compatibility and security. On the one hand there are mandatory and optional ciphers defined in a few RFCs, on the other hand there are clients and servers only implementing subsets of the specification.

Straight forward, the authors wanted strong ciphers, forward secrecy [4] and the best client compatibility possible while still ensuring a cipher string that can be used on legacy installations (e.g. OpenSSL 0.9.8).

Our recommended cipher strings are meant to be used via copy and paste and need to work "out of the box".

- TLSv1.2 is preferred over TLSv1.0 (while still providing a useable cipher string for TLSv1.0 servers).
- AES256 and CAMELLIA256 count as very strong ciphers at the moment.

[4] http://nmav.gnutls.org/2011/12/price-to-pay-for-perfect-forward.html

- AES128 and CAMELLIA128 count as strong ciphers at the moment
- DHE or ECDHE for forward secrecy
- RSA as this will fit most of today's setups
- AES256-SHA as a last resort: with this cipher at the end, even server systems with very old OpenSSL versions will work out of the box (version 0.9.8 for example does not provide support for ECC and TLSv1.1 or above).
  Note however that this cipher suite will not provide forward secrecy. It is meant to provide the same client coverage (eg. support Microsoft crypto libraries) on legacy setups.

3.2.4. Compatibility

TODO: write this section. The idea here is to first document which server (and openssl) version we assumed. Once these parameters are fixed, we then list all clients which are supported for Variant A) and B). Therefore we can document compatibilities to some extent. The sysadmin can then choose roughly what he looses or gains by omitting certain cipher suites.

3.2.5. Choosing your own cipher suites

TODO: Adi... you want to describe how to make your own selection of cipher suites here.

Many of the parts in a ciphersuite are interchangeable. Like the key exchange algorithm in this example: `ECDHE-RSA-AES256-GCM-SHA384` and `DHE-RSA-AES256-GCM-SHA384`. To provide a decent level of security, all algorithms need to be safe (subject to the disclaimer in section 1.4).

Note: There are some very weak cipher suites in every crypto library, most of them for historic reasons or due to legacy standards. The crypto export embargo is a good example [Wik13c]. For the following chapter support of these low-security algorithms is disabled by setting `!EXP:!LOW:!NULL` as part of the cipher string.

TODO: Team: do we need references for all cipher suites considered weak?

Key Exchange

Many algorithms allow secure key exchange. Those are RSA, DH, EDH, ECDSA, ECDH, EECDH amongst others. During the key exchange, keys used for authentication and symmetric encryption are exchanged. For RSA, DSA and ECDSA those keys are derived from the server's public key.

TODO: explain this section

	Key	EC	ephemeral
RSA	RSA	no	no
DH	RSA	no	no
EDH	RSA	no	yes
ECDH	both	yes	no
EECDH	both	yes	yes
DSA	DSA	no	no
ECDSA	DSA	yes	no

Ephemeral Key Exchange uses different keys for authentication (the server's RSA key) and encryption (a randomly created key). This advantage is called "Forward Secrecy" and means that even recorded traffic cannot be decrypted later when someone obtains the server key.
All ephemeral key exchange schemes are based on the Diffie-Hellman algorithm and require pregenerated Diffie-Hellman parameter (which allow fast ephemeral key generation). It is important to note that the Diffie-Hellman parameter settings need to reflect at least the security (speaking in number of bits) as the RSA host key. TODO: add reference!

Elliptic Curves (see section 3.5) required by current TLS standards only consist of the so-called NIST-curves (`secp256r1` and `secp384r1`) which may be weak because the parameters that led to their generation were not properly explained by the authors [DJB13]. Disabling support for Elliptic Curves leads to no ephemeral key exchange being available for the Windows platform. When you decide to use Elliptic Curves despite the uncertainty, make sure to at least use the stronger curve of the two supported by all clients (`secp384r1`).

Other key exchange mechanisms like Pre-Shared Key (PSK) are irrelevant for regular SSL/TLS use.

Authentication

RSA, DSA, DSS, ECDSA, ECDH

During Key Exchange the server proved that he is in control of the private key associated with a certain public key (the server's certificate). The client verifies the server's identity by comparing the signature on the certificate and matching it with its trust database. For details about the trust model of SSL/TLS please see 3.8.

In addition to the server providing its identity, a client might do so as well. That way mutual trust can be established. Another mechanism providing client authentication is Secure Remote Password (SRP)TODO: reference. All those mechanisms require special configuration.

Other authentication mechanisms like Pre Shared Keys are not used in SSL/TLS. Anonymous sessions will not be discussed in this paper.

```
!PSK:!aNULL
```

Encryption

AES, CAMELLIA, SEED, ARIA(?), FORTEZZA(?)...

Other ciphers like IDEA, RC2, RC4, 3DES or DES are weak and therefore not recommended: `!DES:!3DES:!RC2:!RC4:!eNULL`

Message authentication

SHA-1 (SHA), SHA-2 (SHA256, SHA384), AEAD

Note that SHA-1 is considered broken and should not be used. SHA-1 is however the only still available message authentication mechanism supporting TLS1.0/SSLv3. Without SHA-1 most clients will be locked out.

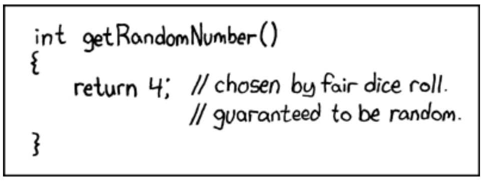
Other hash functions like MD2, MD4 or MD5 are unsafe and broken: `!MD2:!MD4:!MD5`

Combining cipher strings

TODO: Adi... The text below was simply the old text, still left here for reference.

3.3. Random Number Generators

> *"The generation of random numbers is too important to be left to chance."*
>
> — Robert R. Coveyou



```
int getRandomNumber()
{
    return 4;  // chosen by fair dice roll.
               // guaranteed to be random.
}
```

Figure 3.2.: *xkcd, source: http://imgs.xkcd.com/comics/random_number.png, license: CC-BY-NC*

A good source of random numbers is essential for many crypto operations. The key feature of a good random number generator is the non-predictability of the generated numbers. This means that hardware support for generating entropy is essential.

Hardware random number generators in operating systems or standalone components collect entropy from various random events mostly by using the (low bits of the) time an event occurs as an entropy source. The entropy is merged into an entropy pool and in some implementations there is some bookkeeping about the number of random bits available.

3.3.1. When random number generators fail

Random number generators can fail – returning predictable non-random numbers – if not enough entropy is available when random numbers should be generated.

This typically occurs for embedded devices and virtual machines. Embedded devices lack some entropy sources other devices have, e.g.:

- No persistent clock, so boot-time is not contributing to the initial RNG state
- No hard-disk: No entropy from hard-disk timing, no way to store entropy between reboots

Virtual machines emulate some hardware components so that the generated entropy is overestimated. The most critical component that has been shown to return wrong results in an emulated environment is the timing source [Eng11, POL11].

Typically the most vulnerable time where low-entropy situations occur is shortly after a reboot. Unfortunately many operating system installers create cryptographic keys shortly after a reboot [HDWH12].

Another problem is that OpenSSL seeds its internal random generator only seldomly from the hardware random number generator of the operating system. This can lead to situations where a daemon that is started at a time when entropy is low keeps this low-entropy situation for hours leading to predictable session keys [HDWH12].

3.3.2. Linux

TODO: Other architectures, BSD, Windows?

On Linux there are two devices that return random bytes when read; the `/dev/random` can block until sufficient entropy has been collected while `/dev/urandom` will not block and return whatever (possibly insufficient) entropy has been collected so far.

Unfortunately most crypto implementations are using `/dev/urandom` and can produce predictable random numbers if not enough entropy has been collected [HDWH12].

Linux supports the injection of additional entropy into the entropy pool via the device `/dev/random`. On the one hand this is used for keeping entropy across reboots by storing output of /dev/random into a file before shutdown and re-injecting the contents during the boot process. On the other hand this can be used for running a secondary entropy collector to inject entropy into the kernel entropy pool.

On Linux you can check how much entropy is available with the command:

```
$ cat /proc/sys/kernel/random/entropy_avail
```

3.3.3. Recommendations

To avoid situations where a newly deployed server doesn't have enough entropy it is recommended to generate keys (e.g. for SSL or SSH) on a system with a sufficient amount of entropy available and transfer the generated keys to the server. This is especially advisable for small embedded devices or virtual machines.

For embedded devices and virtual machines deploying additional userspace software that generates entropy and feeds this to kernel entropy pool (e.g. by writing to `/dev/random` on Linux) is recommended. Note that only a process with root rights can update the entropy counters in the kernel; non-root or user processes can still feed entropy to the pool but cannot update the counters [Wik13a].

For Linux the `haveged` implementation [HAV13a] based on the HAVEGE [SS03] strong random number generator currently looks like the best choice. It can feed its generated entropy into the kernel entropy pool and recently has grown a mechanism to monitor the quality of generated random numbers [HAV13b]. The memory footprint may be too high for small embedded devices, though.

For systems where – during the lifetime of the keys – it is expected that low-entropy situations occur, RSA keys should be preferred over DSA keys: For DSA, if there is ever insufficient entropy at the time keys are used for signing this may lead to repeated ephemeral keys. An attacker who can guess an ephemeral private key used in such a signature can compromise the DSA secret key. For RSA this can lead to discovery of encrypted plaintext or forged signatures but not to the compromise of the secret key [HDWH12].

3.4. Keylengths

> *"On the choice between AES256 and AES128: I would never consider using AES256, just like I don't wear a helmet when I sit inside my car. It's too much bother for the epsilon improvement in security."*
> — Vincent Rijmen in a personal mail exchange Dec 2013

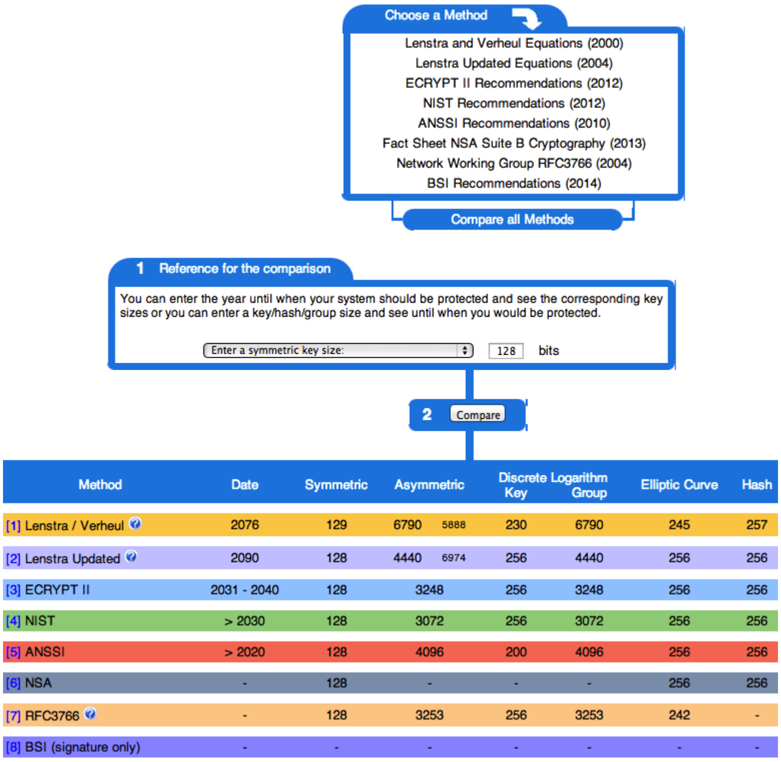
Recommendations on keylengths need to be adapted regularly. Since this document first of all is static and second of all, does not consider itself to be authoritative on keylengths, we would rather refer to existing publications and websites. Recommending a safe key length is a hit-and-miss issue.

Furthermore, when choosing an encryption algorithm and keylength, the designer/sysadmin always needs to consider the value of the information and how long it must be protected. In other words: consider the number of years the data needs to stay confidential.

The ECRYPT II publication [IS12] gives a fascinating overview of strengths of symmetric keys in chapter 5 and chapter 7. Summarizing ECRYPT II, we recommend 128 bit of key strength for symmetric keys. In ECRYPT II, this is considered safe for security level 7, long term protection.

In the same ECRYPT II publication you can find a practical comparison of key size equivalence between symmetric key sizes and RSA, discrete log (DLOG) and EC keylengths. ECRYPT II arrives at the interesting conclusion that for an equivalence of 128 bit symmetric size, you will need to use an 3248 bit RSA key [IS12, chapter 7, page 30].

There are a couple of other studies comparing keylengths and their respective strengths. The website http://www.keylength.com/ compares these papers and offers a good overview of approximations for key lengths based on recommendations by different standardization bodies and academic publications. Figure 3.3 shows a typical comparison of keylengths on this web site.



Choose a Method

Lenstra and Verheul Equations (2000)
Lenstra Updated Equations (2004)
ECRYPT II Recommendations (2012)
NIST Recommendations (2012)
ANSSI Recommendations (2010)
Fact Sheet NSA Suite B Cryptography (2013)
Network Working Group RFC3766 (2004)
BSI Recommendations (2014)

Compare all Methods

1 Reference for the comparison

You can enter the year until when your system should be protected and see the corresponding key sizes or you can enter a key/hash/group size and see until when you would be protected.

Enter a symmetric key size: 128 bits

2 Compare

Method	Date	Symmetric	Asymmetric	Discrete Logarithm		Elliptic Curve	Hash
				Key	Group		
[1] Lenstra / Verheul	2076	129	6790 5888	230	6790	245	257
[2] Lenstra Updated	2090	128	4440 6974	256	4440	256	256
[3] ECRYPT II	2031 - 2040	128	3248	256	3248	256	256
[4] NIST	> 2030	128	3072	256	3072	256	256
[5] ANSSI	> 2020	128	4096	200	4096	256	256
[6] NSA	-	128	-	-	-	256	256
[7] RFC3766	-	128	3253	256	3253	242	-
[8] BSI (signature only)	-	-	-	-	-	-	-

Figure 3.3.: *Screenshot of http://www.keylength.com for 128 bit symmetric key size equivalents*

Summary

- For traditional asymmetric public-key cryptography we consider any key length below 2048 bits to be deprecated at the time of this writing (for long term protection).
- For elliptic curve cryptography we consider key lengths below 256 bits to be inadequate for long term protection.
- For symmetric algorithms we consider anything below 128 bits to be inadequate for long term protection.

Special remark on 3DES

We want to note that 3DES theoretically has 168 bits of security, however based on the NIST Special

Publication 800-57 [5], it is clear that 3DES can only be considered to provide for 80 bits / 112 bits security.

3.5. A note on Elliptic Curve Cryptography

> *"Everyone knows what a curve is, until he has studied enough mathematics to become confused through the countless number of possible exceptions."*
>
> — Felix Klein

Elliptic Curve Cryptography (simply called ECC from now on) is a branch of cryptography that emerged in the mid-1980s. The security of the RSA algorithm is based on the assumption that factoring large numbers is infeasible. Likewise, the security of ECC, DH and DSA is based on the discrete logarithm problem [Wik13b, McC90, Wol13]. Finding the discrete logarithm of an elliptic curve from its public base point is thought to be infeasible. This is known as the Elliptic Curve Discrete Logarithm Problem (ECDLP). ECC and the underlying mathematical foundation are not easy to understand - luckily, there have been some great introductions on the topic lately [6] [7] [8]. ECC provides for much stronger security with less computationally expensive operations in comparison to traditional asymmetric algorithms (See the Section 3.4). The security of ECC relies on the elliptic curves and curve points chosen as parameters for the algorithm in question. Well before the NSA-leak scandal there has been a lot of discussion regarding these parameters and their potential subversion. A part of the discussion involved recommended sets of curves and curve points chosen by different standardization bodies such as the National Institute of Standards and Technology (NIST) [9] which were later widely implemented in most common crypto libraries. Those parameters came under question repeatedly from cryptographers [BL13, Sch13b, W.13]. At the time of writing, there is ongoing research as to the security of various ECC parameters [DJB13]. Most software configured to rely on ECC (be it client or server) is not able to promote or black-list certain curves. It is the hope of the authors that such functionality will be deployed widely soon. The authors of this paper include configurations and recommendations with and without ECC - the reader may choose to adopt those settings as he finds best suited to his environment. The authors will not make this decision for the reader.

A word of warning: One should get familiar with ECC, different curves and parameters if one chooses to adopt ECC configurations. Since there is much discussion on the security of ECC, flawed settings might very well compromise the security of the entire system!

3.6. A note on SHA-1

In the last years several weaknesses have been shown for SHA-1. In particular, collisions on SHA-1 can be found using 2^{63} operations, and recent results even indicate a lower complexity. Therefore, ECRYPT II and NIST recommend against using SHA-1 for generating digital signatures and for other

[5] http://csrc.nist.gov/publications/PubsSPs.html#800-57-part1, pages 63 and 64
[6] http://arstechnica.com/security/2013/10/a-relatively-easy-to-understand-primer-on-elliptic-curve-cryptography
[7] https://www.imperialviolet.org/2010/12/04/ecc.html
[8] http://www.isg.rhul.ac.uk/~sdg/ecc.html
[9] http://www.nist.gov

applications that require collision resistance. The use of SHA-1 in message authentication, e.g. HMAC, is not immediately threatened.

We recommend using SHA-2 whenever available. Since SHA-2 is not supported by older versions of TLS, SHA-1 can be used for message authentication if a higher compatibility with a more diverse set of clients is needed.

Our configurations A and B reflect this. While configuration A does not include SHA-1, configuration B does and thus is more compatible with a wider range of clients.

3.7. A note on Diffie Hellman Key Exchanges

A common question is which Diffie Hellman (DH) Parameters should be used for Diffie Hellman key exchanges[10]. We follow the recommendations in ECRYPT II [IS12, chapter 16]

Where configurable, we recommend using the Diffie Hellman groups defined for IKE, specifically groups 14-18 (2048–8192 bit MODP [KK03]). These groups have been checked by many eyes and can be assumed to be secure.

For convenience, we provide these parameters as PEM files. TODO: put them on the server and insert URL here.

3.8. Public Key Infrastructures

Public-Key Infrastructures try to solve the problem of verifying whether a public key belongs to a given entity, so as to prevent Man In The Middle attacks.

There are two approaches to achieve that: *Certificate Authorities* and the *Web of Trust*.

Certificate Authorities (CAs) sign end-entities' certificates, thereby associating some kind of identity (e.g. a domain name or an email address) with a public key. CAs are used with TLS and S/MIME certificates, and the CA system has a big list of possible and real problems which are summarized in section 3.8.2 and [DKBH13].

The Web of Trust is a decentralized system where people sign each others keys, so that there is a high chance that there is a "trust path" from one key to another. This is used with PGP keys, and while it avoids most of the problems of the CA system, it is more cumbersome.

As alternatives to these public systems, there are two more choices: running a private CA, and manually trusting keys (as it is used with SSH keys or manually trusted keys in web browsers).

The first part of this section addresses how to obtain a certificate in the CA system. The second part offers recommendations on how to improve the security of your PKI.

[10] http://crypto.stackexchange.com/questions/1963/how-large-should-a-diffie-hellman-p-be

3.8.1. Certificate Authorities

In order to get a certificate, you can find an external CA willing to issue a certificate for you, run your own CA, or use self-signed certificates. As always, there are advantages and disadvantages for every one of these options; a balance of security versus usability needs to be found.

Certificates From an External Certificate Authority

There is a fairly large number of commercial CAs that will issue certificates for money. Some of the most ubiquitous commercial CAs are Verisign, GoDaddy, and Teletrust. However, there are also CAs that offer certificates for free. The most notable examples are StartSSL, which is a company that offers some types of certificates for free, and CAcert, which is a non-profit volunteer-based organization that does not charge at all for issuing certificates. Finally, in the research and education field, a number of CAs exist that are generally well-known and well-accepted within the higher-education community.

A large number of CAs is pre-installed in client software's or operating system's"trust stores"; depending on your application, you have to select your CA according to this, or have a mechanism to distribute the chosen CA's root certificate to the clients.

When requesting a certificate from a CA, it is vital that you generate the key pair yourself. In particular, the private key should never be known to the CA. If a CA offers to generate the key pair for you, you should not trust that CA.

Generating a key pair and a certificate request can be done with a number of tools. On Unix-like systems, it is likely that the OpenSSL suite is available to you. In this case, you can generate a private key and a corresponding certificate request as follows:

```
% openssl req -new -nodes -keyout <servername>.key -out <servername>.↩
    csr -newkey rsa:<keysize>
Country Name (2 letter code) [AU]:DE
State or Province Name (full name) [Some-State]:Bavaria
Locality Name (eg, city) []:Munich
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Example
Organizational Unit Name (eg, section) []:Example Section
Common Name (e.g. server FQDN or YOUR name) []:example.com
Email Address []:admin@example.com

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
```

Setting Up Your Own Certificate Authority

In some situations it is advisable to run your own certificate authority. Whether this is a good idea depends on the exact circumstances. Generally speaking, the more centralized the control of the systems in your environment, the fewer pains you will have to go through to deploy your own CA.

On the other hand, running your own CA maximizes the trust level that you can achieve because it minimizes external trust dependencies.

Again using OpenSSL as an example, you can set up your own CA with the following commands on a Debian system:

```
% cd /usr/lib/ssl/misc
% sudo ./CA.pl -newca
```

Answer the questions according to your setup. Now that you have configured your basic settings and issued a new root certificate, you can issue new certificates as follows:

```
% cd /usr/lib/ssl/misc
% sudo ./CA.pl -newreq
```

Alternatively, software such as TinyCA [Wik13e] that acts as a “wrapper” around OpenSSL and tries to make life easier is available.

Creating a Self-Signed Certificate

If the desired trust level is very high and the number of systems involved is limited, the easiest way to set up a secure environment may be to use self-signed certificates. A self-signed certificate is not issued by any CA at all, but is signed by the entity that it is issued to. Thus, the organizational overhead of running a CA is eliminated at the expense of having to establish all trust relationships between entities manually.

With OpenSSL, a self-signed certificate may be created with this command:

```
% openssl req -new -x509 -key privkey.pem -out cacert.pem -days 1095
```

The resulting certificate will by default not be trusted by anyone at all, so in order to be useful, the certificate will have to be made known a priori to all parties that may encounter it.

3.8.2. Hardening PKI

In recent years several CAs were compromised by attackers in order to get a hold of trusted certificates for malicious activities. In 2011 the Dutch CA Diginotar was hacked and all certificates were revoked [Eli11]. Recently Google found certificates issued to them, which were not used by the company [Dam11]. The concept of PKIs heavily depends on the security of CAs. If they get compromised the whole PKI system will fail. Some CAs tend to incorrectly issue certificates that were designated to do a different job than what they were intended to by the CA [Ada13b].

Therefore several security enhancements were introduced by different organizations and vendors [H. 13]. Currently two methods are used, DANE [HS12] and Certificate Pinning [C. 13]. Google recently proposed a new system to detect malicious CAs and certificates called Certificate Transparency [Ada13a].

A. Tools

This section lists tools for checking the security settings.

A.1. SSL & TLS

Server checks via the web

- ssllabs.com offers a great way to check your webserver for misconfigurations. See https://www.ssllabs.com/ssltest/. Furthermore, ssllabs.com has a good best practices tutorial, which focuses on avoiding the most common mistakes in SSL.
- SSL Server certificate installation issues http://www.sslshopper.com/ssl-checker.html
- Check SPDY protocol support and basic TLS setup http://spdycheck.org/
- XMPP/Jabber Server check (Client-to-Server and Server-to-Server) https://xmpp.net/
- Luxsci SMTP TLS Checker https://luxsci.com/extranet/tlschecker.html
- http://checktls.com is a tool for testing arbitrary TLS services.
- TLS and SSH key check https://factorable.net/keycheck.html
- http://tls.secg.org is a tool for testing interoperability of HTTPS implementations for ECC cipher suites.

Browser checks

- Check your browser's SSL capabilities: https://cc.dcsec.uni-hannover.de/ and https://www.ssllabs.com/ssltest/viewMyClient.html.

Command line tools

- http://sourceforge.net/projects/sslscan connects to a given SSL service and shows the cipher suites that are offered.
- http://checktls.com is a tool for testing arbitrary TLS services.
- http://www.bolet.org/TestSSLServer/ tests for BEAST and CRIME vulnerabilities.
- https://github.com/iSECPartners/sslyze Fast and full-featured SSL scanner

- http://nmap.org/ nmap security scanner
- http://www.openssl.net OpenSSL s_client

A.2. Keylength

- http://www.keylength.com comprehensive online resource for comparison of keylengths according to common recommendations and standards in cryptography.

A.3. RNGs

- ENT is a pseudo random number generator sequence tester.
- HaveGE is a tool which increases the Entropy of the Linux random number generator devices. It is based on the HAVEGE algorithm. http://dl.acm.org/citation.cfm?id=945516
- Dieharder a random number generator testing tool.
- CAcert Random another random number generator testing service.

A.4. Guides

- See: https://www.ssllabs.com/downloads/SSL_TLS_Deployment_Best_Practices_1.3.pdf.

B. Links

- IANA official list of Transport Layer Security (TLS) Parameters: https://www.iana.org/assignments/tls-parameters/tls-parameters.txt
- SSL cipher settings: http://www.skytale.net/blog/archives/22-SSL-cipher-setting.html
- Elliptic curves and their implementation (04 Dec 2010): https://www.imperialviolet.org/2010/12/04/ecc.html
- A (relatively easy to understand) primer on elliptic curve cryptography: http://arstechnica.com/security/2013/10/a-relatively-easy-to-understand-primer-on-elliptic-curve-cryptography
- Duraconf, A collection of hardened configuration files for SSL/TLS services (Jacob Appelbaum's github): https://github.com/ioerror/duraconf
- Attacks on SSL a comprehensive study of BEAST, CRIME, TIME, BREACH, LUCKY 13 & RC4 Biases: https://www.isecpartners.com/media/106031/ssl_attacks_survey.pdf
- EFF How to deploy HTTPS correctly: https://www.eff.org/https-everywhere/deploying-https
- Bruce Almighty: Schneier preaches security to Linux faithful (on not recommending to use Blowfish anymore in favor of Twofish): https://www.computerworld.com.au/article/46254/bruce_almighty_schneier_preaches_security_linux_faithful/?pp=3
- Implement FIPS 183-3 for DSA keys (1024bit constraint): https://bugzilla.mindrot.org/show_bug.cgi?id=1647
- Elliptic Curve Cryptography in Practice: http://eprint.iacr.org/2013/734.pdf
- Factoring as a Service: http://crypto.2013.rump.cr.yp.to/981774ce07e51813fd4466612a78601b.pdf
- Black Ops of TCP/IP 2012: http://dankaminsky.com/2012/08/06/bo2012/
- SSL and the Future of Authenticity, Moxie Marlinspike - Black Hat USA 2011: http://www.youtube.com/watch?v=Z7Wl2FW2TcA
- ENISA - Algorithms, Key Sizes and Parameters Report (Oct.'13) http://www.enisa.europa.eu/activities/identity-and-trust/library/deliverables/algorithms-key-sizes-and-parameters-report
- Diffie-Hellman Groups http://ibm.co/18lslZf
- Diffie-Hellman Groups standardized in RFC3526 [KK03] http://datatracker.ietf.org/doc/rfc3526/
- ECC-enabled GnuPG per RFC6637 [Jiv12] https://code.google.com/p/gnupg-ecc

- TLS Security (Survey + Lucky13 + RC4 Attack) by Kenny Paterson https://www.cosic.esat.kuleuven.be/ecc2013/files/kenny.pdf
- Ensuring High-Quality Randomness in Cryptographic Key Generation http://arxiv.org/abs/1309.7366v1
- Wikipedia: Ciphertext Stealing http://en.wikipedia.org/wiki/Ciphertext_stealing
- Wikipedia: Malleability (Cryptography) http://en.wikipedia.org/wiki/Malleability_(cryptography)
- Ritter's Crypto Glossary and Dictionary of Technical Cryptography http://www.ciphersbyritter.com/GLOSSARY.HTM

C. Suggested Reading

This section contains suggested reading material.

- Cryptography Engineering: Design Principles and Practical Applications, Ferguson, N. and Schneier, B. and Kohno, T. (ISBN-13: 978-0470474242)
- Security Engineering: A Guide to Building Dependable Distributed Systems, Anderson, R.J. (ISBN-13: 978-0470068526)
- Applied cryptography: protocols, algorithms, and source code in C, Schneier, B. (ISBN-13: 978-0471117094)
- Guide to Elliptic Curve Cryptography, Hankerson, D. and Vanstone, S. and Menezes, A.J. (ISBN-13: 978-0387952734)
- A Introduction To The Theory of Numbers, Godfrey Harold Hardy, E. M. Wrigh (ISBN-13: 978-0199219865)
- Malicious Cryptography: Exposing Cryptovirology, Young A., Yung, M. (ISBN-13: 978-0764549755)

D. Cipher Suite Name Cross-Reference

This table shows the cipher suite names as IANA defined them, the names OpenSSL uses, and the respective codes.

The list of IANA cipher suite names was retrieved from https://www.iana.org/assignments/tls-parameters/tls-parameters-4.csv on Tue Dec 17 00:18:51 2013.

The list of OpenSSL Ciphers was generated with OpenSSL 1.0.1e 11 Feb 2013.

Code	IANA Name	OpenSSL Name
0x00,0x00	TLS_NULL_WITH_NULL_NULL	
0x00,0x01	TLS_RSA_WITH_NULL_MD5	NULL-MD5
0x00,0x02	TLS_RSA_WITH_NULL_SHA	NULL-SHA
0x00,0x03	TLS_RSA_EXPORT_WITH_RC4_40_MD5	EXP-RC4-MD5
0x00,0x04	TLS_RSA_WITH_RC4_128_MD5	RC4-MD5
0x00,0x05	TLS_RSA_WITH_RC4_128_SHA	RC4-SHA
0x00,0x06	TLS_RSA_EXPORT_WITH_RC2_CBC_40_MD5	EXP-RC2-CBC-MD5
0x00,0x07	TLS_RSA_WITH_IDEA_CBC_SHA	
0x00,0x08	TLS_RSA_EXPORT_WITH_DES40_CBC_SHA	EXP-DES-CBC-SHA
0x00,0x09	TLS_RSA_WITH_DES_CBC_SHA	DES-CBC-SHA
0x00,0x0A	TLS_RSA_WITH_3DES_EDE_CBC_SHA	DES-CBC3-SHA
0x00,0x0B	TLS_DH_DSS_EXPORT_WITH_DES40_CBC_SHA	
0x00,0x0C	TLS_DH_DSS_WITH_DES_CBC_SHA	
0x00,0x0D	TLS_DH_DSS_WITH_3DES_EDE_CBC_SHA	
0x00,0x0E	TLS_DH_RSA_EXPORT_WITH_DES40_CBC_SHA	
0x00,0x0F	TLS_DH_RSA_WITH_DES_CBC_SHA	
0x00,0x10	TLS_DH_RSA_WITH_3DES_EDE_CBC_SHA	
0x00,0x11	TLS_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA	EXP-EDH-DSS-DES-CBC-SHA
0x00,0x12	TLS_DHE_DSS_WITH_DES_CBC_SHA	EDH-DSS-DES-CBC-SHA
0x00,0x13	TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA	EDH-DSS-DES-CBC3-SHA
0x00,0x14	TLS_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA	EXP-EDH-RSA-DES-CBC-SHA
0x00,0x15	TLS_DHE_RSA_WITH_DES_CBC_SHA	EDH-RSA-DES-CBC-SHA
0x00,0x16	TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA	EDH-RSA-DES-CBC3-SHA
0x00,0x17	TLS_DH_anon_EXPORT_WITH_RC4_40_MD5	EXP-ADH-RC4-MD5
0x00,0x18	TLS_DH_anon_WITH_RC4_128_MD5	ADH-RC4-MD5
0x00,0x19	TLS_DH_anon_EXPORT_WITH_DES40_CBC_SHA	EXP-ADH-DES-CBC-SHA
0x00,0x1A	TLS_DH_anon_WITH_DES_CBC_SHA	ADH-DES-CBC-SHA

Code	IANA Name	OpenSSL Name
0x00,0x1B	TLS_DH_anon_WITH_3DES_EDE_CBC_SHA	ADH-DES-CBC3-SHA
0x00,0x1E	TLS_KRB5_WITH_DES_CBC_SHA	
0x00,0x1F	TLS_KRB5_WITH_3DES_EDE_CBC_SHA	
0x00,0x20	TLS_KRB5_WITH_RC4_128_SHA	
0x00,0x21	TLS_KRB5_WITH_IDEA_CBC_SHA	
0x00,0x22	TLS_KRB5_WITH_DES_CBC_MD5	
0x00,0x23	TLS_KRB5_WITH_3DES_EDE_CBC_MD5	
0x00,0x24	TLS_KRB5_WITH_RC4_128_MD5	
0x00,0x25	TLS_KRB5_WITH_IDEA_CBC_MD5	
0x00,0x26	TLS_KRB5_EXPORT_WITH_DES_CBC_40_SHA	
0x00,0x27	TLS_KRB5_EXPORT_WITH_RC2_CBC_40_SHA	
0x00,0x28	TLS_KRB5_EXPORT_WITH_RC4_40_SHA	
0x00,0x29	TLS_KRB5_EXPORT_WITH_DES_CBC_40_MD5	
0x00,0x2A	TLS_KRB5_EXPORT_WITH_RC2_CBC_40_MD5	
0x00,0x2B	TLS_KRB5_EXPORT_WITH_RC4_40_MD5	
0x00,0x2C	TLS_PSK_WITH_NULL_SHA	
0x00,0x2D	TLS_DHE_PSK_WITH_NULL_SHA	
0x00,0x2E	TLS_RSA_PSK_WITH_NULL_SHA	
0x00,0x2F	TLS_RSA_WITH_AES_128_CBC_SHA	AES128-SHA
0x00,0x30	TLS_DH_DSS_WITH_AES_128_CBC_SHA	
0x00,0x31	TLS_DH_RSA_WITH_AES_128_CBC_SHA	
0x00,0x32	TLS_DHE_DSS_WITH_AES_128_CBC_SHA	DHE-DSS-AES128-SHA
0x00,0x33	TLS_DHE_RSA_WITH_AES_128_CBC_SHA	DHE-RSA-AES128-SHA
0x00,0x34	TLS_DH_anon_WITH_AES_128_CBC_SHA	ADH-AES128-SHA
0x00,0x35	TLS_RSA_WITH_AES_256_CBC_SHA	AES256-SHA
0x00,0x36	TLS_DH_DSS_WITH_AES_256_CBC_SHA	
0x00,0x37	TLS_DH_RSA_WITH_AES_256_CBC_SHA	
0x00,0x38	TLS_DHE_DSS_WITH_AES_256_CBC_SHA	DHE-DSS-AES256-SHA
0x00,0x39	TLS_DHE_RSA_WITH_AES_256_CBC_SHA	DHE-RSA-AES256-SHA
0x00,0x3A	TLS_DH_anon_WITH_AES_256_CBC_SHA	ADH-AES256-SHA
0x00,0x3B	TLS_RSA_WITH_NULL_SHA256	NULL-SHA256
0x00,0x3C	TLS_RSA_WITH_AES_128_CBC_SHA256	AES128-SHA256
0x00,0x3D	TLS_RSA_WITH_AES_256_CBC_SHA256	AES256-SHA256
0x00,0x3E	TLS_DH_DSS_WITH_AES_128_CBC_SHA256	
0x00,0x3F	TLS_DH_RSA_WITH_AES_128_CBC_SHA256	
0x00,0x40	TLS_DHE_DSS_WITH_AES_128_CBC_SHA256	DHE-DSS-AES128-SHA256
0x00,0x41	TLS_RSA_WITH_CAMELLIA_128_CBC_SHA	CAMELLIA128-SHA
0x00,0x42	TLS_DH_DSS_WITH_CAMELLIA_128_CBC_SHA	
0x00,0x43	TLS_DH_RSA_WITH_CAMELLIA_128_CBC_SHA	
0x00,0x44	TLS_DHE_DSS_WITH_CAMELLIA_128_CBC_SHA	DHE-DSS-CAMELLIA128-SHA

Code	IANA Name	OpenSSL Name
0x00,0x45	TLS_DHE_RSA_WITH_CAMELLIA_128_CBC_SHA	DHE-RSA-CAMELLIA128-SHA
0x00,0x46	TLS_DH_anon_WITH_CAMELLIA_128_CBC_SHA	ADH-CAMELLIA128-SHA
0x00,0x67	TLS_DHE_RSA_WITH_AES_128_CBC_SHA256	DHE-RSA-AES128-SHA256
0x00,0x68	TLS_DH_DSS_WITH_AES_256_CBC_SHA256	
0x00,0x69	TLS_DH_RSA_WITH_AES_256_CBC_SHA256	
0x00,0x6A	TLS_DHE_DSS_WITH_AES_256_CBC_SHA256	DHE-DSS-AES256-SHA256
0x00,0x6B	TLS_DHE_RSA_WITH_AES_256_CBC_SHA256	DHE-RSA-AES256-SHA256
0x00,0x6C	TLS_DH_anon_WITH_AES_128_CBC_SHA256	ADH-AES128-SHA256
0x00,0x6D	TLS_DH_anon_WITH_AES_256_CBC_SHA256	ADH-AES256-SHA256
0x00,0x84	TLS_RSA_WITH_CAMELLIA_256_CBC_SHA	CAMELLIA256-SHA
0x00,0x85	TLS_DH_DSS_WITH_CAMELLIA_256_CBC_SHA	
0x00,0x86	TLS_DH_RSA_WITH_CAMELLIA_256_CBC_SHA	
0x00,0x87	TLS_DHE_DSS_WITH_CAMELLIA_256_CBC_SHA	DHE-DSS-CAMELLIA256-SHA
0x00,0x88	TLS_DHE_RSA_WITH_CAMELLIA_256_CBC_SHA	DHE-RSA-CAMELLIA256-SHA
0x00,0x89	TLS_DH_anon_WITH_CAMELLIA_256_CBC_SHA	ADH-CAMELLIA256-SHA
0x00,0x8A	TLS_PSK_WITH_RC4_128_SHA	PSK-RC4-SHA
0x00,0x8B	TLS_PSK_WITH_3DES_EDE_CBC_SHA	PSK-3DES-EDE-CBC-SHA
0x00,0x8C	TLS_PSK_WITH_AES_128_CBC_SHA	PSK-AES128-CBC-SHA
0x00,0x8D	TLS_PSK_WITH_AES_256_CBC_SHA	PSK-AES256-CBC-SHA
0x00,0x8E	TLS_DHE_PSK_WITH_RC4_128_SHA	
0x00,0x8F	TLS_DHE_PSK_WITH_3DES_EDE_CBC_SHA	
0x00,0x90	TLS_DHE_PSK_WITH_AES_128_CBC_SHA	
0x00,0x91	TLS_DHE_PSK_WITH_AES_256_CBC_SHA	
0x00,0x92	TLS_RSA_PSK_WITH_RC4_128_SHA	
0x00,0x93	TLS_RSA_PSK_WITH_3DES_EDE_CBC_SHA	
0x00,0x94	TLS_RSA_PSK_WITH_AES_128_CBC_SHA	
0x00,0x95	TLS_RSA_PSK_WITH_AES_256_CBC_SHA	
0x00,0x96	TLS_RSA_WITH_SEED_CBC_SHA	SEED-SHA
0x00,0x97	TLS_DH_DSS_WITH_SEED_CBC_SHA	
0x00,0x98	TLS_DH_RSA_WITH_SEED_CBC_SHA	
0x00,0x99	TLS_DHE_DSS_WITH_SEED_CBC_SHA	DHE-DSS-SEED-SHA
0x00,0x9A	TLS_DHE_RSA_WITH_SEED_CBC_SHA	DHE-RSA-SEED-SHA
0x00,0x9B	TLS_DH_anon_WITH_SEED_CBC_SHA	ADH-SEED-SHA
0x00,0x9C	TLS_RSA_WITH_AES_128_GCM_SHA256	AES128-GCM-SHA256
0x00,0x9D	TLS_RSA_WITH_AES_256_GCM_SHA384	AES256-GCM-SHA384
0x00,0x9E	TLS_DHE_RSA_WITH_AES_128_GCM_SHA256	DHE-RSA-AES128-GCM-SHA256
0x00,0x9F	TLS_DHE_RSA_WITH_AES_256_GCM_SHA384	DHE-RSA-AES256-GCM-SHA384
0x00,0xA0	TLS_DH_RSA_WITH_AES_128_GCM_SHA256	
0x00,0xA1	TLS_DH_RSA_WITH_AES_256_GCM_SHA384	
0x00,0xA2	TLS_DHE_DSS_WITH_AES_128_GCM_SHA256	DHE-DSS-AES128-GCM-SHA256

Code	IANA Name	OpenSSL Name
0x00,0xA3	TLS_DHE_DSS_WITH_AES_256_GCM_SHA384	DHE-DSS-AES256-GCM-SHA384
0x00,0xA4	TLS_DH_DSS_WITH_AES_128_GCM_SHA256	
0x00,0xA5	TLS_DH_DSS_WITH_AES_256_GCM_SHA384	
0x00,0xA6	TLS_DH_anon_WITH_AES_128_GCM_SHA256	ADH-AES128-GCM-SHA256
0x00,0xA7	TLS_DH_anon_WITH_AES_256_GCM_SHA384	ADH-AES256-GCM-SHA384
0x00,0xA8	TLS_PSK_WITH_AES_128_GCM_SHA256	
0x00,0xA9	TLS_PSK_WITH_AES_256_GCM_SHA384	
0x00,0xAA	TLS_DHE_PSK_WITH_AES_128_GCM_SHA256	
0x00,0xAB	TLS_DHE_PSK_WITH_AES_256_GCM_SHA384	
0x00,0xAC	TLS_RSA_PSK_WITH_AES_128_GCM_SHA256	
0x00,0xAD	TLS_RSA_PSK_WITH_AES_256_GCM_SHA384	
0x00,0xAE	TLS_PSK_WITH_AES_128_CBC_SHA256	
0x00,0xAF	TLS_PSK_WITH_AES_256_CBC_SHA384	
0x00,0xB0	TLS_PSK_WITH_NULL_SHA256	
0x00,0xB1	TLS_PSK_WITH_NULL_SHA384	
0x00,0xB2	TLS_DHE_PSK_WITH_AES_128_CBC_SHA256	
0x00,0xB3	TLS_DHE_PSK_WITH_AES_256_CBC_SHA384	
0x00,0xB4	TLS_DHE_PSK_WITH_NULL_SHA256	
0x00,0xB5	TLS_DHE_PSK_WITH_NULL_SHA384	
0x00,0xB6	TLS_RSA_PSK_WITH_AES_128_CBC_SHA256	
0x00,0xB7	TLS_RSA_PSK_WITH_AES_256_CBC_SHA384	
0x00,0xB8	TLS_RSA_PSK_WITH_NULL_SHA256	
0x00,0xB9	TLS_RSA_PSK_WITH_NULL_SHA384	
0x00,0xBA	TLS_RSA_WITH_CAMELLIA_128_CBC_SHA256	
0x00,0xBB	TLS_DH_DSS_WITH_CAMELLIA_128_CBC_SHA256	
0x00,0xBC	TLS_DH_RSA_WITH_CAMELLIA_128_CBC_SHA256	
0x00,0xBD	TLS_DHE_DSS_WITH_CAMELLIA_128_CBC_SHA256	
0x00,0xBE	TLS_DHE_RSA_WITH_CAMELLIA_128_CBC_SHA256	
0x00,0xBF	TLS_DH_anon_WITH_CAMELLIA_128_CBC_SHA256	
0x00,0xC0	TLS_RSA_WITH_CAMELLIA_256_CBC_SHA256	
0x00,0xC1	TLS_DH_DSS_WITH_CAMELLIA_256_CBC_SHA256	
0x00,0xC2	TLS_DH_RSA_WITH_CAMELLIA_256_CBC_SHA256	
0x00,0xC3	TLS_DHE_DSS_WITH_CAMELLIA_256_CBC_SHA256	
0x00,0xC4	TLS_DHE_RSA_WITH_CAMELLIA_256_CBC_SHA256	
0x00,0xC5	TLS_DH_anon_WITH_CAMELLIA_256_CBC_SHA256	
0x00,0xFF	TLS_EMPTY_RENEGOTIATION_INFO_SCSV	
0xC0,0x01	TLS_ECDH_ECDSA_WITH_NULL_SHA	ECDH-ECDSA-NULL-SHA
0xC0,0x02	TLS_ECDH_ECDSA_WITH_RC4_128_SHA	ECDH-ECDSA-RC4-SHA
0xC0,0x03	TLS_ECDH_ECDSA_WITH_3DES_EDE_CBC_SHA	ECDH-ECDSA-DES-CBC3-SHA
0xC0,0x04	TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA	ECDH-ECDSA-AES128-SHA

Code	IANA Name	OpenSSL Name
0xC0,0x05	TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA	ECDH-ECDSA-AES256-SHA
0xC0,0x06	TLS_ECDHE_ECDSA_WITH_NULL_SHA	ECDHE-ECDSA-NULL-SHA
0xC0,0x07	TLS_ECDHE_ECDSA_WITH_RC4_128_SHA	ECDHE-ECDSA-RC4-SHA
0xC0,0x08	TLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA	ECDHE-ECDSA-DES-CBC3-SHA
0xC0,0x09	TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA	ECDHE-ECDSA-AES128-SHA
0xC0,0x0A	TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA	ECDHE-ECDSA-AES256-SHA
0xC0,0x0B	TLS_ECDH_RSA_WITH_NULL_SHA	ECDH-RSA-NULL-SHA
0xC0,0x0C	TLS_ECDH_RSA_WITH_RC4_128_SHA	ECDH-RSA-RC4-SHA
0xC0,0x0D	TLS_ECDH_RSA_WITH_3DES_EDE_CBC_SHA	ECDH-RSA-DES-CBC3-SHA
0xC0,0x0E	TLS_ECDH_RSA_WITH_AES_128_CBC_SHA	ECDH-RSA-AES128-SHA
0xC0,0x0F	TLS_ECDH_RSA_WITH_AES_256_CBC_SHA	ECDH-RSA-AES256-SHA
0xC0,0x10	TLS_ECDHE_RSA_WITH_NULL_SHA	ECDHE-RSA-NULL-SHA
0xC0,0x11	TLS_ECDHE_RSA_WITH_RC4_128_SHA	ECDHE-RSA-RC4-SHA
0xC0,0x12	TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA	ECDHE-RSA-DES-CBC3-SHA
0xC0,0x13	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA	ECDHE-RSA-AES128-SHA
0xC0,0x14	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA	ECDHE-RSA-AES256-SHA
0xC0,0x15	TLS_ECDH_anon_WITH_NULL_SHA	AECDH-NULL-SHA
0xC0,0x16	TLS_ECDH_anon_WITH_RC4_128_SHA	AECDH-RC4-SHA
0xC0,0x17	TLS_ECDH_anon_WITH_3DES_EDE_CBC_SHA	AECDH-DES-CBC3-SHA
0xC0,0x18	TLS_ECDH_anon_WITH_AES_128_CBC_SHA	AECDH-AES128-SHA
0xC0,0x19	TLS_ECDH_anon_WITH_AES_256_CBC_SHA	AECDH-AES256-SHA
0xC0,0x1A	TLS_SRP_SHA_WITH_3DES_EDE_CBC_SHA	SRP-3DES-EDE-CBC-SHA
0xC0,0x1B	TLS_SRP_SHA_RSA_WITH_3DES_EDE_CBC_SHA	SRP-RSA-3DES-EDE-CBC-SHA
0xC0,0x1C	TLS_SRP_SHA_DSS_WITH_3DES_EDE_CBC_SHA	SRP-DSS-3DES-EDE-CBC-SHA
0xC0,0x1D	TLS_SRP_SHA_WITH_AES_128_CBC_SHA	SRP-AES-128-CBC-SHA
0xC0,0x1E	TLS_SRP_SHA_RSA_WITH_AES_128_CBC_SHA	SRP-RSA-AES-128-CBC-SHA
0xC0,0x1F	TLS_SRP_SHA_DSS_WITH_AES_128_CBC_SHA	SRP-DSS-AES-128-CBC-SHA
0xC0,0x20	TLS_SRP_SHA_WITH_AES_256_CBC_SHA	SRP-AES-256-CBC-SHA
0xC0,0x21	TLS_SRP_SHA_RSA_WITH_AES_256_CBC_SHA	SRP-RSA-AES-256-CBC-SHA
0xC0,0x22	TLS_SRP_SHA_DSS_WITH_AES_256_CBC_SHA	SRP-DSS-AES-256-CBC-SHA
0xC0,0x23	TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256	ECDHE-ECDSA-AES128-SHA256
0xC0,0x24	TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384	ECDHE-ECDSA-AES256-SHA384
0xC0,0x25	TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA256	ECDH-ECDSA-AES128-SHA256
0xC0,0x26	TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA384	ECDH-ECDSA-AES256-SHA384
0xC0,0x27	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256	ECDHE-RSA-AES128-SHA256
0xC0,0x28	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384	ECDHE-RSA-AES256-SHA384
0xC0,0x29	TLS_ECDH_RSA_WITH_AES_128_CBC_SHA256	ECDH-RSA-AES128-SHA256
0xC0,0x2A	TLS_ECDH_RSA_WITH_AES_256_CBC_SHA384	ECDH-RSA-AES256-SHA384
0xC0,0x2B	TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256	ECDHE-ECDSA-AES128-GCM-SHA256
0xC0,0x2C	TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384	ECDHE-ECDSA-AES256-GCM-SHA384

Code	IANA Name	OpenSSL Name
0xC0,0x2D	TLS_ECDH_ECDSA_WITH_AES_128_GCM_SHA256	ECDH-ECDSA-AES128-GCM-SHA256
0xC0,0x2E	TLS_ECDH_ECDSA_WITH_AES_256_GCM_SHA384	ECDH-ECDSA-AES256-GCM-SHA384
0xC0,0x2F	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	ECDHE-RSA-AES128-GCM-SHA256
0xC0,0x30	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	ECDHE-RSA-AES256-GCM-SHA384
0xC0,0x31	TLS_ECDH_RSA_WITH_AES_128_GCM_SHA256	ECDH-RSA-AES128-GCM-SHA256
0xC0,0x32	TLS_ECDH_RSA_WITH_AES_256_GCM_SHA384	ECDH-RSA-AES256-GCM-SHA384
0xC0,0x33	TLS_ECDHE_PSK_WITH_RC4_128_SHA	
0xC0,0x34	TLS_ECDHE_PSK_WITH_3DES_EDE_CBC_SHA	
0xC0,0x35	TLS_ECDHE_PSK_WITH_AES_128_CBC_SHA	
0xC0,0x36	TLS_ECDHE_PSK_WITH_AES_256_CBC_SHA	
0xC0,0x37	TLS_ECDHE_PSK_WITH_AES_128_CBC_SHA256	
0xC0,0x38	TLS_ECDHE_PSK_WITH_AES_256_CBC_SHA384	
0xC0,0x39	TLS_ECDHE_PSK_WITH_NULL_SHA	
0xC0,0x3A	TLS_ECDHE_PSK_WITH_NULL_SHA256	
0xC0,0x3B	TLS_ECDHE_PSK_WITH_NULL_SHA384	
0xC0,0x3C	TLS_RSA_WITH_ARIA_128_CBC_SHA256	
0xC0,0x3D	TLS_RSA_WITH_ARIA_256_CBC_SHA384	
0xC0,0x3E	TLS_DH_DSS_WITH_ARIA_128_CBC_SHA256	
0xC0,0x3F	TLS_DH_DSS_WITH_ARIA_256_CBC_SHA384	
0xC0,0x40	TLS_DH_RSA_WITH_ARIA_128_CBC_SHA256	
0xC0,0x41	TLS_DH_RSA_WITH_ARIA_256_CBC_SHA384	
0xC0,0x42	TLS_DHE_DSS_WITH_ARIA_128_CBC_SHA256	
0xC0,0x43	TLS_DHE_DSS_WITH_ARIA_256_CBC_SHA384	
0xC0,0x44	TLS_DHE_RSA_WITH_ARIA_128_CBC_SHA256	
0xC0,0x45	TLS_DHE_RSA_WITH_ARIA_256_CBC_SHA384	
0xC0,0x46	TLS_DH_anon_WITH_ARIA_128_CBC_SHA256	
0xC0,0x47	TLS_DH_anon_WITH_ARIA_256_CBC_SHA384	
0xC0,0x48	TLS_ECDHE_ECDSA_WITH_ARIA_128_CBC_SHA256	
0xC0,0x49	TLS_ECDHE_ECDSA_WITH_ARIA_256_CBC_SHA384	
0xC0,0x4A	TLS_ECDH_ECDSA_WITH_ARIA_128_CBC_SHA256	
0xC0,0x4B	TLS_ECDH_ECDSA_WITH_ARIA_256_CBC_SHA384	
0xC0,0x4C	TLS_ECDHE_RSA_WITH_ARIA_128_CBC_SHA256	
0xC0,0x4D	TLS_ECDHE_RSA_WITH_ARIA_256_CBC_SHA384	
0xC0,0x4E	TLS_ECDH_RSA_WITH_ARIA_128_CBC_SHA256	
0xC0,0x4F	TLS_ECDH_RSA_WITH_ARIA_256_CBC_SHA384	
0xC0,0x50	TLS_RSA_WITH_ARIA_128_GCM_SHA256	
0xC0,0x51	TLS_RSA_WITH_ARIA_256_GCM_SHA384	
0xC0,0x52	TLS_DHE_RSA_WITH_ARIA_128_GCM_SHA256	
0xC0,0x53	TLS_DHE_RSA_WITH_ARIA_256_GCM_SHA384	
0xC0,0x54	TLS_DH_RSA_WITH_ARIA_128_GCM_SHA256	

Code	IANA Name	OpenSSL Name
0xC0,0x55	TLS_DH_RSA_WITH_ARIA_256_GCM_SHA384	
0xC0,0x56	TLS_DHE_DSS_WITH_ARIA_128_GCM_SHA256	
0xC0,0x57	TLS_DHE_DSS_WITH_ARIA_256_GCM_SHA384	
0xC0,0x58	TLS_DH_DSS_WITH_ARIA_128_GCM_SHA256	
0xC0,0x59	TLS_DH_DSS_WITH_ARIA_256_GCM_SHA384	
0xC0,0x5A	TLS_DH_anon_WITH_ARIA_128_GCM_SHA256	
0xC0,0x5B	TLS_DH_anon_WITH_ARIA_256_GCM_SHA384	
0xC0,0x5C	TLS_ECDHE_ECDSA_WITH_ARIA_128_GCM_SHA256	
0xC0,0x5D	TLS_ECDHE_ECDSA_WITH_ARIA_256_GCM_SHA384	
0xC0,0x5E	TLS_ECDH_ECDSA_WITH_ARIA_128_GCM_SHA256	
0xC0,0x5F	TLS_ECDH_ECDSA_WITH_ARIA_256_GCM_SHA384	
0xC0,0x60	TLS_ECDHE_RSA_WITH_ARIA_128_GCM_SHA256	
0xC0,0x61	TLS_ECDHE_RSA_WITH_ARIA_256_GCM_SHA384	
0xC0,0x62	TLS_ECDH_RSA_WITH_ARIA_128_GCM_SHA256	
0xC0,0x63	TLS_ECDH_RSA_WITH_ARIA_256_GCM_SHA384	
0xC0,0x64	TLS_PSK_WITH_ARIA_128_CBC_SHA256	
0xC0,0x65	TLS_PSK_WITH_ARIA_256_CBC_SHA384	
0xC0,0x66	TLS_DHE_PSK_WITH_ARIA_128_CBC_SHA256	
0xC0,0x67	TLS_DHE_PSK_WITH_ARIA_256_CBC_SHA384	
0xC0,0x68	TLS_RSA_PSK_WITH_ARIA_128_CBC_SHA256	
0xC0,0x69	TLS_RSA_PSK_WITH_ARIA_256_CBC_SHA384	
0xC0,0x6A	TLS_PSK_WITH_ARIA_128_GCM_SHA256	
0xC0,0x6B	TLS_PSK_WITH_ARIA_256_GCM_SHA384	
0xC0,0x6C	TLS_DHE_PSK_WITH_ARIA_128_GCM_SHA256	
0xC0,0x6D	TLS_DHE_PSK_WITH_ARIA_256_GCM_SHA384	
0xC0,0x6E	TLS_RSA_PSK_WITH_ARIA_128_GCM_SHA256	
0xC0,0x6F	TLS_RSA_PSK_WITH_ARIA_256_GCM_SHA384	
0xC0,0x70	TLS_ECDHE_PSK_WITH_ARIA_128_CBC_SHA256	
0xC0,0x71	TLS_ECDHE_PSK_WITH_ARIA_256_CBC_SHA384	
0xC0,0x72	TLS_ECDHE_ECDSA_WITH_CAMELLIA_128_CBC_SHA256	
0xC0,0x73	TLS_ECDHE_ECDSA_WITH_CAMELLIA_256_CBC_SHA384	
0xC0,0x74	TLS_ECDH_ECDSA_WITH_CAMELLIA_128_CBC_SHA256	
0xC0,0x75	TLS_ECDH_ECDSA_WITH_CAMELLIA_256_CBC_SHA384	
0xC0,0x76	TLS_ECDHE_RSA_WITH_CAMELLIA_128_CBC_SHA256	
0xC0,0x77	TLS_ECDHE_RSA_WITH_CAMELLIA_256_CBC_SHA384	
0xC0,0x78	TLS_ECDH_RSA_WITH_CAMELLIA_128_CBC_SHA256	
0xC0,0x79	TLS_ECDH_RSA_WITH_CAMELLIA_256_CBC_SHA384	
0xC0,0x7A	TLS_RSA_WITH_CAMELLIA_128_GCM_SHA256	
0xC0,0x7B	TLS_RSA_WITH_CAMELLIA_256_GCM_SHA384	
0xC0,0x7C	TLS_DHE_RSA_WITH_CAMELLIA_128_GCM_SHA256	

Code	IANA Name	OpenSSL Name
0xC0,0x7D	TLS_DHE_RSA_WITH_CAMELLIA_256_GCM_SHA384	
0xC0,0x7E	TLS_DH_RSA_WITH_CAMELLIA_128_GCM_SHA256	
0xC0,0x7F	TLS_DH_RSA_WITH_CAMELLIA_256_GCM_SHA384	
0xC0,0x80	TLS_DHE_DSS_WITH_CAMELLIA_128_GCM_SHA256	
0xC0,0x81	TLS_DHE_DSS_WITH_CAMELLIA_256_GCM_SHA384	
0xC0,0x82	TLS_DH_DSS_WITH_CAMELLIA_128_GCM_SHA256	
0xC0,0x83	TLS_DH_DSS_WITH_CAMELLIA_256_GCM_SHA384	
0xC0,0x84	TLS_DH_anon_WITH_CAMELLIA_128_GCM_SHA256	
0xC0,0x85	TLS_DH_anon_WITH_CAMELLIA_256_GCM_SHA384	
0xC0,0x86	TLS_ECDHE_ECDSA_WITH_CAMELLIA_128_GCM_SHA256	
0xC0,0x87	TLS_ECDHE_ECDSA_WITH_CAMELLIA_256_GCM_SHA384	
0xC0,0x88	TLS_ECDH_ECDSA_WITH_CAMELLIA_128_GCM_SHA256	
0xC0,0x89	TLS_ECDH_ECDSA_WITH_CAMELLIA_256_GCM_SHA384	
0xC0,0x8A	TLS_ECDHE_RSA_WITH_CAMELLIA_128_GCM_SHA256	
0xC0,0x8B	TLS_ECDHE_RSA_WITH_CAMELLIA_256_GCM_SHA384	
0xC0,0x8C	TLS_ECDH_RSA_WITH_CAMELLIA_128_GCM_SHA256	
0xC0,0x8D	TLS_ECDH_RSA_WITH_CAMELLIA_256_GCM_SHA384	
0xC0,0x8E	TLS_PSK_WITH_CAMELLIA_128_GCM_SHA256	
0xC0,0x8F	TLS_PSK_WITH_CAMELLIA_256_GCM_SHA384	
0xC0,0x90	TLS_DHE_PSK_WITH_CAMELLIA_128_GCM_SHA256	
0xC0,0x91	TLS_DHE_PSK_WITH_CAMELLIA_256_GCM_SHA384	
0xC0,0x92	TLS_RSA_PSK_WITH_CAMELLIA_128_GCM_SHA256	
0xC0,0x93	TLS_RSA_PSK_WITH_CAMELLIA_256_GCM_SHA384	
0xC0,0x94	TLS_PSK_WITH_CAMELLIA_128_CBC_SHA256	
0xC0,0x95	TLS_PSK_WITH_CAMELLIA_256_CBC_SHA384	
0xC0,0x96	TLS_DHE_PSK_WITH_CAMELLIA_128_CBC_SHA256	
0xC0,0x97	TLS_DHE_PSK_WITH_CAMELLIA_256_CBC_SHA384	
0xC0,0x98	TLS_RSA_PSK_WITH_CAMELLIA_128_CBC_SHA256	
0xC0,0x99	TLS_RSA_PSK_WITH_CAMELLIA_256_CBC_SHA384	
0xC0,0x9A	TLS_ECDHE_PSK_WITH_CAMELLIA_128_CBC_SHA256	
0xC0,0x9B	TLS_ECDHE_PSK_WITH_CAMELLIA_256_CBC_SHA384	
0xC0,0x9C	TLS_RSA_WITH_AES_128_CCM	
0xC0,0x9D	TLS_RSA_WITH_AES_256_CCM	
0xC0,0x9E	TLS_DHE_RSA_WITH_AES_128_CCM	
0xC0,0x9F	TLS_DHE_RSA_WITH_AES_256_CCM	
0xC0,0xA0	TLS_RSA_WITH_AES_128_CCM_8	
0xC0,0xA1	TLS_RSA_WITH_AES_256_CCM_8	
0xC0,0xA2	TLS_DHE_RSA_WITH_AES_128_CCM_8	
0xC0,0xA3	TLS_DHE_RSA_WITH_AES_256_CCM_8	
0xC0,0xA4	TLS_PSK_WITH_AES_128_CCM	

Code	IANA Name	OpenSSL Name
0xC0,0xA5	TLS_PSK_WITH_AES_256_CCM	
0xC0,0xA6	TLS_DHE_PSK_WITH_AES_128_CCM	
0xC0,0xA7	TLS_DHE_PSK_WITH_AES_256_CCM	
0xC0,0xA8	TLS_PSK_WITH_AES_128_CCM_8	
0xC0,0xA9	TLS_PSK_WITH_AES_256_CCM_8	
0xC0,0xAA	TLS_PSK_DHE_WITH_AES_128_CCM_8	
0xC0,0xAB	TLS_PSK_DHE_WITH_AES_256_CCM_8	

E. Further research

The following is a list of services, software packages, hardware devices or protocols that we considered documenting but either did not manage to document yet or might be able to document later. We encourage input from the Internet community.

- whatsapp (might be problematic
  since a user/admin can't change anything)
- Lync
- Skype (might be problematic since a user/admin can't change anything)
- Wi-Fi APs, 802.1X
- Tomcat
- SIP
- SRTP
- DNSSec (mention BCPs)
- DANE
- TOR
- S/Mime (check are there any BCPs?)
- TrueCrypt, LUKS, FileVault
- AFS
- Kerberos
- NNTP
- NTPs tlsdate
- BGP / OSPF
- SILC
- LDAP
- seclayer-tcp
- Commerical network equipment vendors
- RADIUS
- Moxa , APC, und co... ICS . Ethernet to serial
- telnet (only sensible recommendation: *DON't!!*)
- rsyslog
- v6 spoofing (look at work by Ferndo Gont, Marc Heuse, et. al.)
- tinc
- racoon
- l2tp
- rsync
- telnets
- ftps
- webmin (probably the same recommendations as with Apache apply, but where does that need to be configured?)
- plesk (same as webmin)
- phpmyadmin (same as webmin)
- DSL modems (where to start?)
- UPnP, natPmp
- SAML federated auth providers [1]
- Microsoft SQL Server

[1] e.g., all the REFEDS folks (https://refeds.org/)), including InCommon (http://www.incommon.org/federation/metadata.html https://wiki.shibboleth.net/confluence/display/SHIB2/TrustManagement

Bibliography

[Ada13a] Adam Langley, Ben Laurie, Emilia Kasper. Certificate Transparency. http://www.certificate-transparency.org http://datatracker.ietf.org/doc/rfc6962/, 07 2013.

[Ada13b] Adam Langley, et. al. Go X.509 Verification Source Code. https://code.google.com/p/go/source/browse/src/pkg/crypto/x509/verify.go#173, 12 2013.

[And08] Ross Anderson. *Security engineering*. Wiley.com, 2008. http://www.cl.cam.ac.uk/~rja14/book.html

[BL13] D. J. Bernstein and Tanja Lange. Security dangers of the NIST curves. Presentation slides, September 2013. http://cr.yp.to/talks/2013.09.16/slides-djb-20130916-a4.pdf

[C. 13] C. Evans and C. Palmer. Public Key Pinning Extension for HTTP. http://tools.ietf.org/html/draft-ietf-websec-key-pinning-09, November 2013.

[Dam11] Damon Poeter. Fake Google Certificate Puts Gmail at Risk. http://www.pcmag.com/article2/0,2817,2392063,00.asp, August 2011.

[DJB13] Safecurves: choosing safe curves for elliptic-curve cryptography. Technical background, December 2013. Accessed 2013-12-09. http://safecurves.cr.yp.to/rigid.html

[DKBH13] Zakir Durumeric, James Kasten, Michael Bailey, and J. Alex Halderman. Analysis of the HTTPS certificate ecosystem. In *Proceedings of the 13th Internet Measurement Conference*, October 2013. https://jhalderm.com/pub/papers/https-imc13.pdf

[Eli11] Elinor Mills. Fraudulent Google certificate points to Internet attack. http://news.cnet.com/8301-27080_3-20098894-245/fraudulent-google-certificate-points-to-internet-attack/, August 2011.

[Eng11] Jakob Engblom. Evaluating HAVEGE randomness. Blog: Observations from uppsala, February 2011. http://jakob.engbloms.se/archives/1374

[ENI13] ENISA and Vincent Rijmen, Nigel P. Smart, Bogdan warinschi, Gaven Watson. Enisa - algorithms, key sizes and parameters report. Technical report, Oct 2013. http://www.enisa.europa.eu/activities/identity-and-trust/library/deliverables/algorithms-key-sizes-and-parameters-report

[fSidIB13] Bundesamt für Sicherheit in der Informationstechnik (BSI). Bsi tr-02102 kryptographische verfahren. Technical report, Jan 2013. https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR02102/BSI-TR-02102_pdf

[Gle13] Glenn Greenwald. Edward Snowden: NSA whistleblower answers reader questions. http://www.theguardian.com/world/2013/jun/17/

edward-snowden-nsa-files-whistleblower, http://www.theguardian.com/world/2013/jun/17/edward-snowden-nsa-files-whistleblower, 07 2013.

[H. 13] H. Tschofenig and E. Lear. Evolving the Web Public Key Infrastructure. http://tools.ietf.org/html/draft-tschofenig-iab-webpki-evolution-01.txt, November 2013.

[HAV13a] haveged – a simple entropy daemon. Software homepage, December 2013. Accessed 2013-12-06. http://www.issihosts.com/haveged/

[HAV13b] haveged – a simple entropy daemon: Runtime testing. Technical background, December 2013. Accessed 2013-12-06. http://www.issihosts.com/haveged/

[HDWH12] Nadia Heninger, Zakir Durumeric, Eric Wustrow, and J. Alex Halderman. Mining your Ps and Qs: Detection of widespread weak keys in network devices. In *Proceedings of the 21st USENIX Security Symposium*, August 2012. https://factorable.net/weakkeys12.extended.pdf

[Hof05] P. Hoffman. Cryptographic Suites for IPsec. RFC 4308 (Proposed Standard), December 2005. http://www.ietf.org/rfc/rfc4308.txt

[HS12] P. Hoffman and J. Schlyter. The DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS) Protocol: TLSA. RFC 6698 (Proposed Standard), August 2012. http://www.ietf.org/rfc/rfc6698.txt

[IS12] ECRYPT II and D SYM. Ecrypt ii. pages 79–86, 2012. http://www.ecrypt.eu.org/documents/D.SPA.20.pdf

[Jiv12] A. Jivsov. Elliptic Curve Cryptography (ECC) in OpenPGP. RFC 6637 (Proposed Standard), June 2012. http://www.ietf.org/rfc/rfc6637.txt

[KBC97] H. Krawczyk, M. Bellare, and R. Canetti. HMAC: Keyed-Hashing for Message Authentication. RFC 2104 (Informational), February 1997. Updated by RFC 6151. http://www.ietf.org/rfc/rfc2104.txt

[KK03] T. Kivinen and M. Kojo. More Modular Exponential (MODP) Diffie-Hellman groups for Internet Key Exchange (IKE). RFC 3526 (Proposed Standard), May 2003. http://www.ietf.org/rfc/rfc3526.txt

[KL08] J. Katz and Y. Lindell. *Introduction to modern cryptography*. Chapman and Hall/CRC Cryptography and Network Security Series. Chapman & Hall/CRC, 2008. http://books.google.at/books?id=WIc_AQAAIAAJ

[LK08] M. Lepinski and S. Kent. Additional Diffie-Hellman Groups for Use with IETF Standards. RFC 5114 (Informational), January 2008. http://www.ietf.org/rfc/rfc5114.txt

[LS11] L. Law and J. Solinas. Suite B Cryptographic Suites for IPsec. RFC 6379 (Informational), October 2011. http://www.ietf.org/rfc/rfc6379.txt

[McC90] Kevin S. McCurley. The discrete logarithm problem. In *Cryptology and Computational Number Theory, Proceedings of Symposia in Applied Mathematics*, volume 42, pages 49–74, 1990. http://www.mccurley.org/papers/dlog.pdf

[POL11] Weak random number generation within virtualized environments. Security Advisory 2011-02, PolarSSL, December 2011. https://polarssl.org/tech-updates/security-advisories/polarssl-security-advisory-2011-02

[Sch13a] Bruce Schneier. The NSA is breaking most encryption on the internet. Blog: Schneier on security, September 2013. https://www.schneier.com/blog/archives/2013/09/the_nsa_is_brea.html

[Sch13b] Bruce Schneier. The NSA is breaking most encryption on the internet. Answer to blog comment, September 2013. https://www.schneier.com/blog/archives/2013/09/the_nsa_is_brea.html#c1675929

[SS03] A. Seznec and N. Sendrier. HAVEGE: a user-level software heuristic for generating empirically strong random numbers. *ACM Transactions on Modeling and Computer Simulation*, 13(4):334–346, October 2003. http://www.irisa.fr/caps/projects/hipsor/scripts/down.php?id=13781296&ext=.pdf

[W.13] D. W. Should we trust the NIST-recommended ECC parameters? Stackexchange question, Stackexchange Q&A Site, September 2013. http://crypto.stackexchange.com/questions/10263/should-we-trust-the-nist-recommended-ecc-parameters

[Wik13a] /dev/random. Wikipedia, Wikipedia, December 2013. Accessed 2013-12-06. http://en.wikipedia.org/wiki/dev/random

[Wik13b] Discrete logarithm. Wikipedia, Wikipedia, December 2013. Accessed 2013-12-12. https://en.wikipedia.org/wiki/Discrete_logarithm

[Wik13c] Export of cryptography in the United States. Wikipedia, Wikipedia, December 2013. Accessed 2013-12-09. http://en.wikipedia.org/wiki/Export_of_cryptography_in_the_United_States

[Wik13d] Tempest (codename). Wikipedia, Wikipedia, December 2013. Accessed 2013-12-12. https://en.wikipedia.org/wiki/Tempest_(codename)

[Wik13e] Tinyca. Wikipedia, Wikipedia, December 2013. Accessed 2013-12-24. http://en.wikipedia.org/wiki/TinyCA

[Wol13] Elliptic curve. Math dictionary entry, Wolfram Research Mathworld, December 2013. Accessed 2013-12-12. http://mathworld.wolfram.com/EllipticCurve.html

[YF13] Yuval Yarom and Katrina Falkner. Flush+ reload: a high resolution, low noise, l3 cache side-channel attack, 2013. http://eprint.iacr.org/2013/448.pdf