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, Alexander Würstlein, Aaron Zauner, Pepi Zawodsky

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

April 25, 2017

Contents

1. Abstract

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.

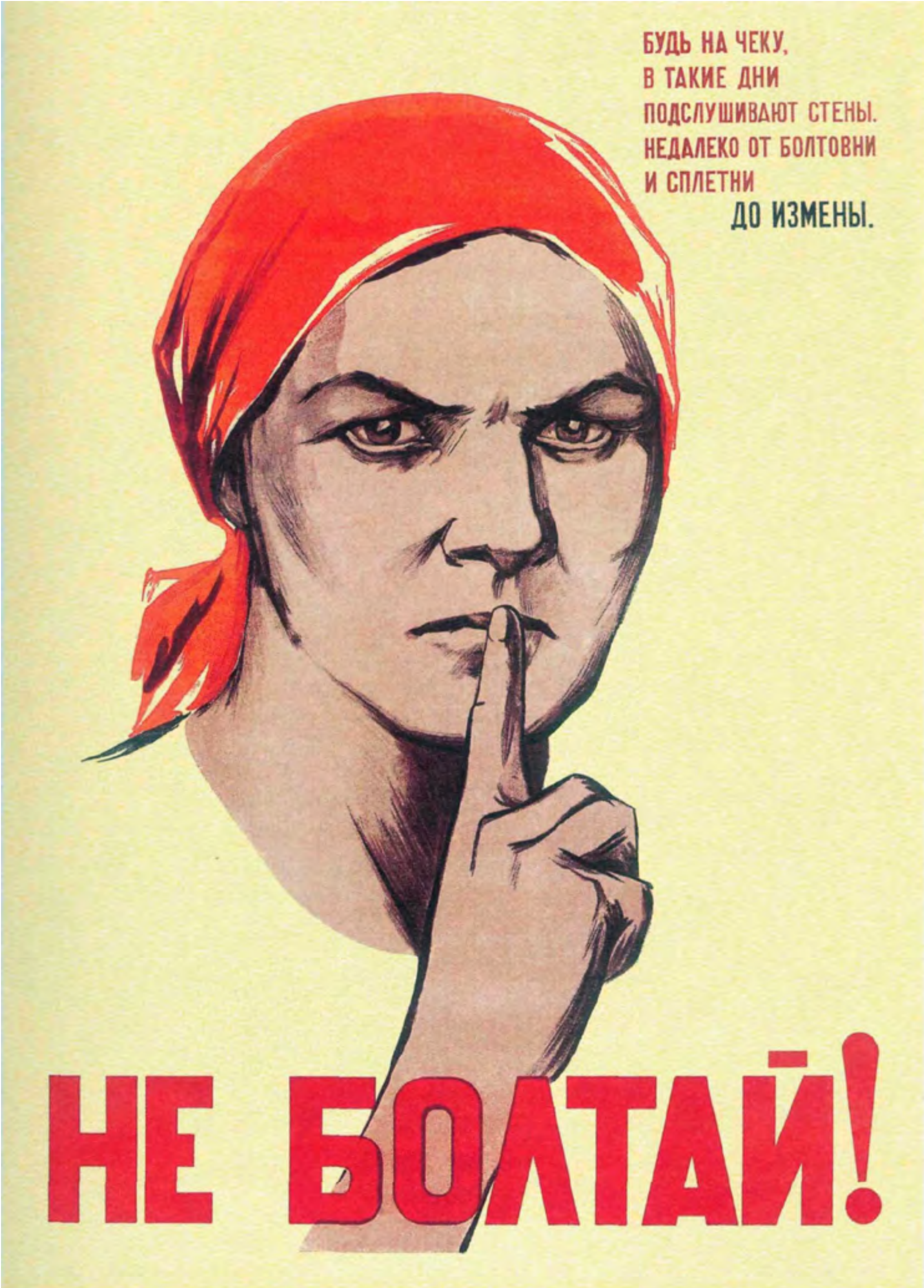
As Schneier noted in , it seems that intelligence agencies and adversaries on the Internet are not breaking so much the mathematics of encryption per se, but rather use software and hardware weaknesses, subvert standardization processes, plant backdoors, rig random number generators and most of all exploit 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 references in appendix [cha:links] and [cha:suggested-reading] 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 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.

1.1. Acknowledgements



Do not talk unencrypted

1.1. 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
- Dirksen-Thedens, Mathis
- Dulaunoy, Alexandre
- Gühring Philipp
- Grigg, Ian
- Haslinger, Gunnar
- Huebl, Axel
- Kovacic, Daniel
- Lenzhofer, Stefan
- Lorünser, Thomas
- Maass, Max
- Mehlmauer, Christian
- Millauer, Tobias
- Mirbach, Andreas
- O'Brien, Hugh
- Pacher, Christoph
- Palfrader, Peter
- Pape, Tobias (layout)
- Petukhova, Anna (Logo)
- Pichler, Patrick
- Riebesel, Nicolas
- Roeckx, Kurt
- Roesen, Jens
- Rublik, Martin
- Schüpany, Mathias
- Schwarz, René («DigNative»)
- Seidl, Eva (PDF layout)
- Van Horenbeeck, Maarten
- 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.

2. Introduction

2.1. Audience

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

2.2. Related publications

Ecrypt II *[3]*, ENISA's report on Algorithms, key sizes and parameters *[14]* and BSI's Technische Richtlinie TR-02102 *[10]* 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.

2.3. How to read this guide

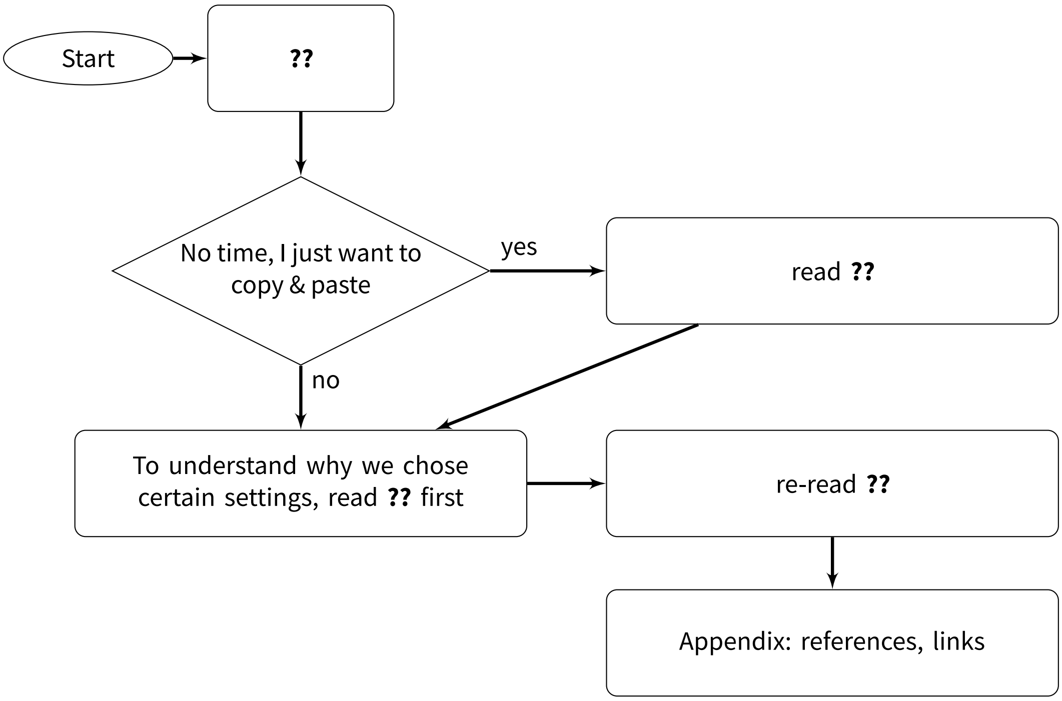
This guide tries to accommodate two needs: first of all, having a handy reference on how to configure the most common services' crypto settings and second of all, explain a bit of background on cryptography. This background is essential if the reader wants to chose his or her own background is essential if the reader wants to choose his or her own cipher string 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 *Practical recommendations* ("Practical recommendations").

It is important to know that in this guide the authors arrived at two recommendations: *Cipher string A* and *Cipher string B*. While the former is a hardened recommendation the latter is a weaker one but provides wider compatibility. *Cipher strings A and B* are described in section-recommendedciphers.

However, for the quick copy & paste approach it is important to know that this guide assumes users are happy with *Cipher string B*.

While chapter *Practical recommendations* is intended to serve as a copy & paste reference, chapter chapter-Theory ("Theory") explains the reasoning behind *cipher string B*. In particular, section section-CipherSuites 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.



2.4. 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 cite{snowdenGuardianGreenwald}

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 Operations Security[1], social engineering attacks, protection against tempest [22] attack techniques, thwarting 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] https://en.wikipedia.org/wiki/Operations_security

[2] An easy to read yet very insightful recent example is the "FLUSH+RELOAD" technique [23] 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 *[4]*.

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 **??**. 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.

2.4.1. 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, industrial control systems, etc.)
- Wireless Access Points

[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

- Smart-cards/chip cards

2.5. 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 "**??**". 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[1] for this paper.

Public peer-review and multiple eyes checking of our guide is the best strategy we can imagine at the present moment[2].

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

[1] https://github.com/BetterCrypto/Applied-Crypto-Hardening
[2] http://www.wired.com/opinion/2013/10/how-to-design-and-defend-against-the-perfect-backdoor/

3. Practical recommendations

3.1. Webservers

3.1.1. Apache

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])

Tested with Versions

- Apache 2.2.22, Debian Wheezy with OpenSSL 1.0.1e
- Apache 2.4.6, Debian Jessie with OpenSSL 1.0.1e
- Apache 2.4.10, Debian Jessie 8.2 with OpenSSL 1.0.1k
- Apache 2.4.7, Ubuntu 14.04.2 Trusty with Openssl 1.0.1f
- Apache 2.4.6, CentOS Linux 7 (Core) with OpenSSL 1.0.1e

Settings

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

Additional settings

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

References

- Apache2 Docs on SSL and TLS: url{https://httpd.apache.org/docs/2.4/ssl/}

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

How to test

See appendix cha-tools

3.1.2. lighttpd

Tested with Versions

- lighttpd/1.4.31-4 with OpenSSL 1.0.1e on Debian Wheezy
- 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)
- lighttpd/1.4.31, Ubuntu 14.04.2 Trusty with Openssl 1.0.1f

Settings

Starting with lighttpd version 1.4.29 Diffie-Hellman and Elliptic-Curve Diffie-Hellman key agreement protocols are supported. By default, elliptic curve "prime256v1" (also "secp256r1") will be used, if no other is given. To select special curves, it is possible to set them using the configuration options `ssl.dh-file` and `ssl.ec-curve`.

Please read section section-DH for more information on Diffie Hellman key exchange and elliptic curves.

Additional settings

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

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). ECDHE 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: url{http://redmine.lighttpd.net/projects/1/wiki/HowToRedirectHttpToHttps}
- Lighttpd Docs SSL: url{http://redmine.lighttpd.net/projects/lighttpd/wiki/Docs_SSL}
- Release 1.4.30 (How to mitigate BEAST attack) url{http://redmine.lighttpd.net/projects/lighttpd/wiki/Release-1_4_30}
- SSL Compression disabled by default: url{http://redmine.lighttpd.net/issues/2445}

How to test

See appendix cha-tools

3.1.3. 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 Wheezy
- 1.4.4 with OpenSSL 1.0.1e on Debian Wheezy
- 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)
- 1.4.6 with OpenSSL 1.0.1f on Ubuntu 14.04.2 LTS

Settings

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 section-DH 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:

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

The variable *$server_name* refers to the first *server_name* entry in your config file. If you specify more than one *server_name* only the first will be taken. Please be sure to not use the *$host* variable here because it contains data controlled by the user.

References

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

How to test

See appendix cha-tools

3.1.4. Cherokee

Tested with Version

- Cherokee/1.2.104 on Debian Wheezy with OpenSSL 1.0.1e 11 Feb 2013

Settings

The configuration of the cherokee webserver is performed by an admin interface available via the web. It then writes the configuration to `/etc/cherokee/cherokee.conf`, the important lines of such a configuration file can be found at the end of this section.

- General Settings
  - Network

    SSL/TLS back-end OpenSSL/libssl
  - Ports to listen

    Port 443

    TLS TLS/SSL port
- Virtual Servers, For each vServer on tab *Security*:
  - *Required SSL/TLS Values*: Fill in the correct paths for *Certificate* and *Certificate key*

- Advanced Options

  Ciphers `|cipherStringB|`

  Server Preference Prefer

  Compression Disabled

- Advanced: TLS

  SSL version 2 and SSL version 3 No

  TLS version 1, TLS version 1.1 and TLS version 1.2 Yes

Additional settings

For each vServer on the Security tab it is possible to set the Diffie Hellman length to up to 4096 bits. We recommend to use >1024 bits. More information about Diffie-Hellman and which curves are recommended can be found in section section-DH.

In Advanced: TLS it is possible to set the path to a Diffie Hellman parameters file for 512, 1024, 2048 and 4096 bits.

HSTS can be configured on host-basis in section *vServers / Security / HTTP Strict Transport Security (HSTS)*:

Enable HSTS Accept

HSTS Max-Age 15768000

Include Subdomains *depends on your setup*

To redirect HTTP to HTTPS, configure a new rule per Virtual Server in the *Behavior* tab. The rule is *SSL/TLS* combined with a *NOT* operator. As *Handler* define *Redirection* and use `/(.*)$` as *Regular Expression* and *https://${host}/$1* as *Substitution*.

References

- Cookbook: SSL, TLS and certificates: url{http://cherokee-project.com/doc/cookbook_ssl.html}
- Cookbook: Redirecting all traffic from HTTP to HTTPS: url{http://cherokee-project.com/doc/cookbook_http_to_https.html}

How to test

See appendix cha-tools

3.1.5. MS IIS

To configure SSL/TLS on Windows Server IIS Crypto can be used. [2] Simply start the Programm, no installation required. The tool changes the registry keys described below. A restart is required for the changes to take effect.

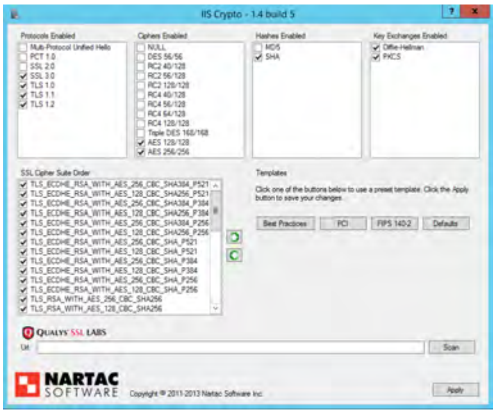


Fig. 3.1.: *IIS Crypto Tool*

Instead of using the IIS Crypto Tool the configuration can be set using the Windows Registry. The following Registry keys apply to the newer Versions of Windows (Windows 7, Windows Server 2008, Windows Server 2008 R2, Windows Server 2012 and Windows Server 2012 R2). For detailed information about the older versions see the Microsoft knowledgebase article.[3]

```
[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\SecurityProviders\Schannel]
[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\SecurityProviders\Schannel\Ciphers]
[HKEY_LOCAL_
↪MACHINE\SYSTEM\CurrentControlSet\Control\SecurityProviders\Schannel\CipherSuites]
[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\SecurityProviders\Schannel\Hashes]
[HKEY_LOCAL_
↪MACHINE\SYSTEM\CurrentControlSet\Control\SecurityProviders\Schannel\KeyExchangeAlgorithms]
[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\SecurityProviders\Schannel\Protocols]
```

Tested with Version

- Windows Server 2008
- Windows Server 2008 R2
- Windows Server 2012
- Windows Server 2012 R2

[2] https://www.nartac.com/Products/IISCrypto/
[3] http://support.microsoft.com/kb/245030/en-us

- Windows Vista and Internet Explorer 7 and upwards
- Windows 7 and Internet Explorer 8 and upwards
- Windows 8 and Internet Explorer 10 and upwards
- Windows 8.1 and Internet Explorer 11

Settings

When trying to avoid RC4 (RC4 biases) as well as CBC (BEAST-Attack) by using GCM and to support perfect forward secrecy, Microsoft SChannel (SSL/TLS, Auth,.. Stack) supports ECDSA but lacks support for RSA signatures (see ECC suite B doubts[4]).

Since one is stuck with ECDSA, an elliptic curve certificate 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[5]
2. Registry[6]
3. IIS Crypto [7]
4. Powershell

Table *Client support* 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 1.2` are turned on.

Table 3.1.: *Client support*

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 *Client support* 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.

[4] http://safecurves.cr.yp.to/rigid.html
[5] http://msdn.microsoft.com/en-us/library/windows/desktop/bb870930(v=vs.85).aspx
[6] http://support.microsoft.com/kb/245030
[7] https://www.nartac.com/Products/IISCrypto/

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

It's recommended to use Strict-Transport-Security: max-age=15768000 for detailed information visit the[8] Microsoft knowledgebase.

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

```
Set-WebConfiguration -Location "$WebSiteName/$WebApplicationName" `
    -Filter 'system.webserver/security/access' `
    -Value "SslRequireCert"
```

Justification for special settings (if needed)

References

- http://support.microsoft.com/kb/245030/en-us
- http://support.microsoft.com/kb/187498/en-us

How to test

See appendix cha-tools

[8] http://www.iis.net/configreference/system.webserver/httpprotocol/customheaders

3.2. SSH

Warning: Please be advised that any change in the SSH-Settings of your server might cause problems connecting to the server or starting/reloading the SSH-Daemon itself. So every time you configure your SSH-Settings on a remote server via SSH itself, ensure that you have a second open connection to the server, which you can use to reset or adapt your changes!

3.2.1. OpenSSH

Tested with Version

- OpenSSH 6.6p1 (Gentoo)
- OpenSSH 6.6p1-2 on Ubuntu 14.04.2 LTS

Settings

Note: OpenSSH 6.6p1 now supports Curve25519

Tested with Version

OpenSSH 6.5 (Debian Jessie)

Settings

Tested with Version

- OpenSSH 6.0p1 (Debian wheezy)

Settings

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 6.6p1). DSA host keys have been removed on purpose, the DSS standard does not support for DSA keys stronger than 1024bit[1] which is far below current

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

standards (see section section-keylengths). 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
```

and observe the key exchange in the output.

3.2.2. Cisco ASA

Tested with Versions

- 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.

3.2.3. Cisco IOS

Tested with Versions

- 15.0
- 15.1
- 15.2

Settings

```
crypto key generate rsa modulus 4096 label SSH-KEYS
ip ssh rsa keypair-name SSH-KEYS
ip ssh version 2
ip ssh dh min size 2048

line vty 0 15
transport input ssh
```

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. From IOS Version 15.0 onwards, 4096 Bit rsa keys are supported and should be used according to the paradigm “use longest supported key”. Also, do not forget to disable telnet vty access.

References

- http://www.cisco.com/en/US/docs/ios/sec_user_services/configuration/guide/sec_cfg_secure_shell.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.

3.3. Mail Servers

This section documents the most common mail servers. Mail servers may usually be grouped into three categories:

- the mail submission agent (MSA)
- the mail transfer agent (MTA)/mail exchanger (MX)
- the mail delivery agent (MDA)

An e-mail client (mail user agent, MUA) submits mail to the MSA. This is usually been done using the Simple Mail Transfer Protocol (SMTP). Afterwards, the mail is transmitted by the MTA over the Internet to the MTA of the receiver. This happens again via SMTP. Finally, the mail client of the receiver will fetch mail from an MDA usually via the Internet Message Access Protocol (IMAP) or the Post Office Protocol (POP).

As MSAs and MTAs both use SMTP as transfer protocols, both functionalities may often be implemented with the same software. On the other hand, MDA software might or might not implement both IMAP and POP.

3.3.1. TLS usage in mail server protocols

E-mail protocols support TLS in two different ways. It may be added as a protocol wrapper on a different port. This method is referred to as Implicit TLS or as protocol variants SMTPS, IMAPS and POP3S. The other method is to establish a cleartext session first and switch to TLS afterwards by issuing the STARTTLS command.

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

3.3.2. Recommended configuration

We recommend to use the following settings for Mail Transfer Agents:

- correctly setup MX, A and PTR RRs without using CNAMEs at all.
- the hostname used as HELO/EHLO in outgoing mail shall match the PTR RR
- enable opportunistic TLS, using the STARTTLS mechanism on port 25
- Implicit TLS on port 465 may be offered additionally

- use server and client certificates (most server certificates are client certificates as well)
- either the common name or at least an alternate subject name of the certificate shall match the PTR RR (client mode) or the MX RR (server mode)
- do not use self signed certificates
- accept all cipher suites, as the alternative would be to fall back to cleartext transmission
- **an execption to the last sentence is that MTAs *MUST NOT* enable SSLv2** protocol support, due to the DROWN attackfootnote{url{https://drownattack.com/drown-attack-paper.pdf}}.

For MSA operation we recommend: * listen on submission port 587 with mandatory STARTTLS * optionally listen on port 465 with Implicit TLS * enforce SMTP AUTH even for local networks * ensure that SMTP AUTH is not allowed on unencrypted connections * only use the recommended cipher suites if all connecting MUAs support them For MDA operation we recommend:

- listen on the protocol port (143 for IMAP, 110 for POP3) with mandatory STARTTLS
- optionally listen on Implicit TLS ports (993 for IMAPS, 995 for POP3S)
- enforce authentication even for local networks
- make sure that authentication is not allowed on unencrypted connections
- use the recommended cipher suites if all connecting MUAs support them
- turn off SSLv2 (DROWN attackfootnote{url{https://drownattack.com/drown-attack-paper.pdf}})

3.3.3. Dovecot

Tested with Version

- Dovecot 2.1.7, Debian Wheezy (without: `ssl\_prefer\_server\_ciphers`)
- Dovecot 2.2.9, Debian Jessie
- Dovecot 2.2.13, Debian 8.2 Jessie
- Dovecot 2.0.19apple1 on OS X Server 10.8.5 (without: `ssl\_prefer\_server\_ciphers}`)
- Dovecot 2.2.9 on Ubuntu 14.04 trusty

Settings

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 < 2.2.14 does not support disabling TLS compression. ≥ 2.2.14[1] use: `ssl_options = no_compression`
- Dovecot < 2.2.7 uses fixed DH parameters. ≥ 2.2.7[2] greater DH-Parameters are supported: `ssl_dh_parameters_length = 2048`.

References

- http://wiki2.dovecot.org/SSL

How to test

```
openssl s_client -crlf -connect SERVER.TLD:993
openssl s_client -crlf -connect SERVER.TLD:995
openssl s_client -crlf -starttls imap -connect SERVER.TLD:143
openssl s_client -crlf -starttls pop3 -connect SERVER.TLD:110
```

SSLyze[3] offers scanning for common vulnerabilities and displays Protocols and Cipher-Suites.

```
sslyze.exe --regular SERVER.TLD:993
sslyze.exe --regular SERVER.TLD:995
sslyze.exe --regular --starttls=imap SERVER.TLD:143
sslyze.exe --regular --starttls=pop3 SERVER.TLD:110
```

3.3.4. cyrus-imapd

Tested with Versions

- 2.4.17

[1] http://www.dovecot.org/doc/NEWS-2.2
[2] http://hg.dovecot.org/dovecot-2.2/rev/43ab5abeb8f0
[3] https://github.com/nabla-c0d3/sslyze/releases

Settings

To activate SSL/TLS configure your certificate with

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

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 This way MUAs can only authenticate with plain text authentication schemes after issuing the STARTTLS command. Providing CRAM-MD5 or DIGEST-MD5 methods is not recommended.

To support POP3/IMAP on ports 110/143 with STARTTLS and POP3S/IMAPS on ports 995/993 check the SERVICES section in `cyrus.conf`

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
```

3.3.5. Postfix

Tested with Versions

- Postfix 2.9.6, Debian Wheezy with OpenSSL 1.0.1e
- Postfix 2.11.0 on Ubuntu 14.04.02 with OpenSSL 1.0.1f

Settings

Postfix has five internal lists of ciphers, and the possibility to switch between those with *smtpd_tls_ciphers*. However, we leave this at its default value for server to server connections, as many mail servers only support outdated protocols and ciphers. We consider bad encryption still better than plain text transmission. For connections to MUAs, TLS is mandatory and the ciphersuite is modified.

MX and SMTP client configuration: As discussed in section subsection-smtp_general, because of opportunistic encryption we do not restrict the list of ciphers or protocols for communication with other mail servers to avoid transmission in plain text. There are still some steps needed to enable TLS, all in `main.cf`:

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

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

For those users who want to use EECDH key exchange, it is possible to customize this via: The default value since Postfix 2.8 is "strong".

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 section-DH) 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
```

3.3.6. Exim

Tested with Versions

- Exim 4.82, Debian Jessie
- Exim 4.82, Ubuntu 14.04.2 with OpenSSL 1.0.1e

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

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

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

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

Add the following rules on top of your acl_smtp_mail: 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: don't forget to add intermediate certificates to the .pem file if needed.

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

Listen on smtp(25) port only

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_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 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_sslv3 +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 [rfc5114] (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
```

3.3.7. Cisco ESA/IronPort

Tested with Version

- AsyncOS 7.6.1
- AsyncOS 8.5.6
- AsyncOS 9.0.0, 9.5.0, 9.6.0, 9.7.0

Settings

Import your certificate(s) using the WEBUI (Network -> Certificates).

From AsyncOS 9.0 and up, SSL parameters for inbound SMTP, outbound SMTP and GUI access can be configured in one step via the WEBUI (System Administration -> SSL Configuration, see figure fig-ach_ironport_ssl_settings on page **??**).

For all versions prior to 9.0, you have to connect to the CLI and configure the SSL parameters separately, as shown below using inbound SMTP as example.

```
ironport.example.com> sslconfig
sslconfig settings:
  GUI HTTPS method:  sslv3tlsv1
  GUI HTTPS ciphers: RC4-SHA:RC4-MD5:ALL
  Inbound SMTP method:  sslv3tlsv1
  Inbound SMTP ciphers: RC4-SHA:RC4-MD5:ALL
  Outbound SMTP method:  sslv3tlsv1
  Outbound SMTP ciphers: RC4-SHA:RC4-MD5:ALL

Choose the operation you want to perform:
- GUI - Edit GUI HTTPS ssl settings.
- INBOUND - Edit Inbound SMTP ssl settings.
- OUTBOUND - Edit Outbound SMTP ssl settings.
- VERIFY - Verify and show ssl cipher list.
[]> inbound

Enter the inbound SMTP ssl method you want to use.
1. SSL v2.
2. SSL v3
3. TLS v1
4. SSL v2 and v3
5. SSL v3 and TLS v1
6. SSL v2, v3 and TLS v1
[5]> 3

Enter the inbound SMTP ssl cipher you want to use.
[RC4-SHA:RC4-MD5:ALL]>
↪EDH+CAMELLIA:EDH+aRSA:EECDH+aRSA+AESGCM:EECDH+aRSA+SHA256:EECDH:+CAMELLIA128:+AES128:+SSLv3:!
↪aNULL:!eNULL:!LOW:!3DES:!MD5:!EXP:!PSK:!DSS:!RC4:!SEED:!IDEA:!ECDSA:kEDH:CAMELLIA128-
↪SHA:AES128-SHA

sslconfig settings:
  GUI HTTPS method:  sslv3tlsv1
  GUI HTTPS ciphers: RC4-SHA:RC4-MD5:ALL
  Inbound SMTP method:  tlsv1
  Inbound SMTP ciphers:
↪EDH+CAMELLIA:EDH+aRSA:EECDH+aRSA+AESGCM:EECDH+aRSA+SHA384:EECDH+aRSA+SHA256:EECDH:+CAMELLIA256:+AES256:+CA
↪aNULL:!eNULL:!LOW:!3DES:!MD5:!EXP:!PSK:!SRP:!DSS:!RC4:!SEED:!ECDSA:CAMELLIA256-SHA:AES256-
↪SHA:CAMELLIA128-SHA:AES128-SHA
  Outbound SMTP method:  sslv3tlsv1
  Outbound SMTP ciphers: RC4-SHA:RC4-MD5:ALL
```

Note that starting with AsyncOS 9.0 SSLv3 is disabled by default, whereas the default cipher set is still `RC4-SHA:RC4-MD5:ALL` (see figure fig-ach_ironport_ssl_settings on page **??**).

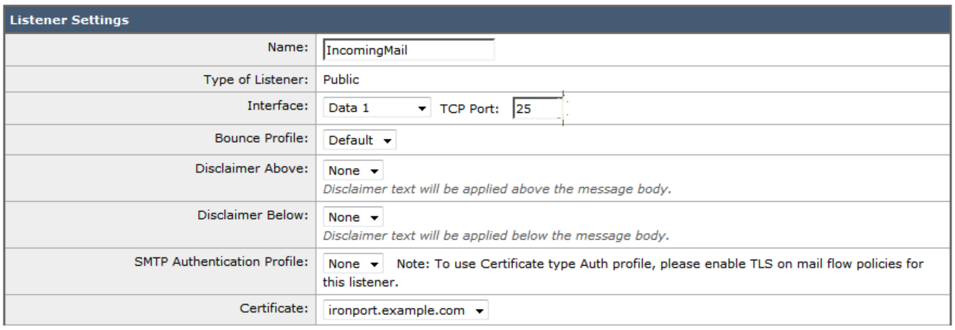
SSL Configuration		
GUI HTTPS:	Methods:	TLS v1
	SSL Cipher(s) to use:	RC4-SHA:RC4-MD5:ALL
Inbound SMTP:	Methods:	TLS v1
	SSL Cipher(s) to use:	RC4-SHA:RC4-MD5:ALL
Outbound SMTP:	Methods:	TLS v1
	SSL Cipher(s) to use:	RC4-SHA:RC4-MD5:ALL
		Edit Settings...

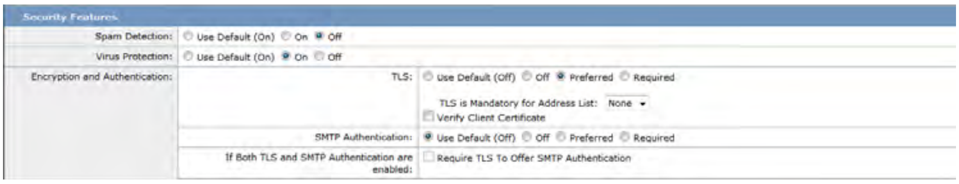
After committing these changes in the CLI, you have to activate the use of TLS in several locations.

For inbound connections, first select the appropriate certificate in the settings of each listener you want to have TLS enabled on (Network -> Listeners, see figure fig-ach_ironport_listener_cert on page **??**). Afterwards, for each listener, configure all Mail Flow Policies which have their Connection Behavior set to “Accept” or “Relay” to at least prefer TLS (Mail Policies -> Mail Flow Policies, see figure fig-ach_ironport_mail_flow_tls on page **??**).

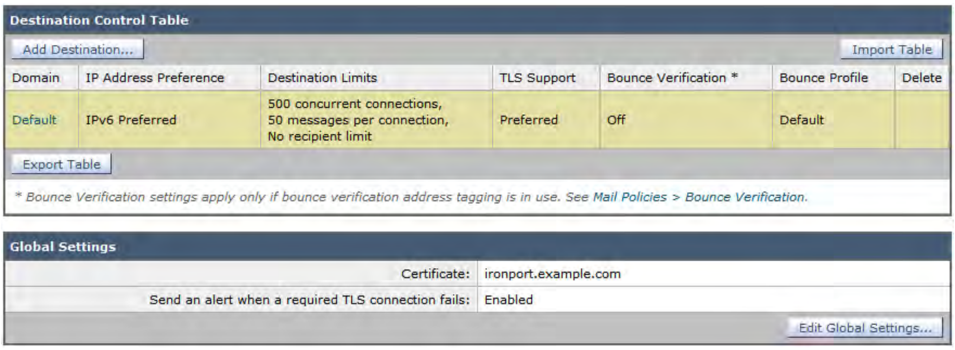
It is recommended to also enable TLS in the default Mail Flow Policy, because these settings will be inherited by newly created policies, unless specifically overwritten.

TLS can be enforced by creating a new Mail Flow Policy with TLS set to “required”, creating a new Sender Group defining the addresses of the sending mail servers for which you want to enforce encryption (Mail Policies -> HAT Overview) and using this new Sender Group in conjunction with the newly created Mail Flow Policy.





TLS settings for outbound connections have to be configured within the Destination Controls (Mail Policies -> Destination Controls). Choose the appropriate SSL certificate within the global settings and configure TLS to be preferred in the default profile to enable it for all outbound connections. After these two steps the Destination Control overview page should look like figure fig-ach_ironport_dest_control on page **??**. To enforce TLS for a specific destination domain, add an entry to the Destination Control Table and set “TLS Support” to “required”.



Limitations

All AsyncOS releases prior to version 9.5 use OpenSSL 0.9.8. Therefore TLS 1.2 is not supported in these versions and some of the suggested ciphers won't work. Starting with AsyncOS 9.5 TLS 1.2 is fully supported.[4] You can check the supported ciphers on the CLI by using the option `verify` from within the `sslconfig` command:

```
[]> verify

 Enter the ssl cipher you want to verify.
 []>
↪EDH+CAMELLIA:EDH+aRSA:EECDH+aRSA+AESGCM:EECDH+aRSA+SHA256:EECDH:+CAMELLIA128:+AES128:+SSLv3:!
↪aNULL:!eNULL:!LOW:!3DES:!MD5:!EXP:!PSK:!DSS:!RC4:!SEED:!IDEA:!ECDSA:kEDH:CAMELLIA128-
↪SHA:AES128-SHA

 DHE-RSA-CAMELLIA256-SHA SSLv3 Kx=DH       Au=RSA  Enc=Camellia(256) Mac=SHA1
 DHE-RSA-CAMELLIA128-SHA SSLv3 Kx=DH       Au=RSA  Enc=Camellia(128) Mac=SHA1
 DHE-RSA-AES256-SHA      SSLv3 Kx=DH       Au=RSA  Enc=AES(256)  Mac=SHA1
 DHE-RSA-AES128-SHA      SSLv3 Kx=DH       Au=RSA  Enc=AES(128)  Mac=SHA1
 CAMELLIA128-SHA         SSLv3 Kx=RSA      Au=RSA  Enc=Camellia(128) Mac=SHA1
 AES128-SHA              SSLv3 Kx=RSA      Au=RSA  Enc=AES(128)  Mac=SHA1
```

How to test

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

3.4. VPNs

3.4.1. 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

[4] http://www.cisco.com/c/dam/en/us/td/docs/security/esa/esa9-5/ESA_9-5_Release_Notes.pdf, Changed Behaviour, page 4

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[1].

For a key composed of upper- and lowercase letters, numbers, and two additional symbols[2], table *PSK lengths* gives the minimum lengths in characters.

Table 3.2.: *PSK lengths*

IKE Hash	PSK length (chars)
SHA256	43
SHA384	64
SHA512	86

Cryptographic Suites: 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.[3]

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 section-recommendedciphers are shown in table *IPSEC Cryptographic Suites*.

Table 3.3.: *IPSEC Cryptographic Suites*

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

Phase 1: Alternatively to the pre-defined cipher suites, you can define your own, as described in this and the next section.

Phase 1 is the mutual authentication and key exchange phase; table *IPSEC Phase 1 parameters* shows the parameters.

Use only "main mode", as "aggressive mode" has known security vulnerabilities[4].

Table 3.4.: *IPSEC Phase 1 parameters*

	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

[1] It is used in a HMAC, see **RFC 2104** and the discussion starting in http://www.vpnc.org/ietf-ipsec/02.ipsec/msg00268.html.
[2] 64 possible values = 6 bits
[3] **RFC 6379**, **RFC 4308**
[4] http://ikecrack.sourceforge.net/

Phase 2: Phase 2 is where the parameters that protect the actual data are negotiated; recommended parameters are shown in table *IPSEC Phase 2 parameters*.

Table 3.5.: *IPSEC Phase 2 parameters*

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

References

- "A Cryptographic Evaluation of IPsec", Niels Ferguson and Bruce Schneier:[12]

3.4.2. Check Point FireWall-1

Tested with Versions

- R77 (should work with any currently supported version)

Settings

Please see section section-IPSECgeneral 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 choose one of the encryption suites in the properties dialog (figure *VPN Community encryption properties*), or proceed to "Custom Encryption...", where you can set encryption and hash for Phase 1 and 2 (figure *Custom Encryption Suite Properties*).

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

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 *Remote Access Encryption Properties*).

[12] https://www.schneier.com/paper-ipsec.pdf

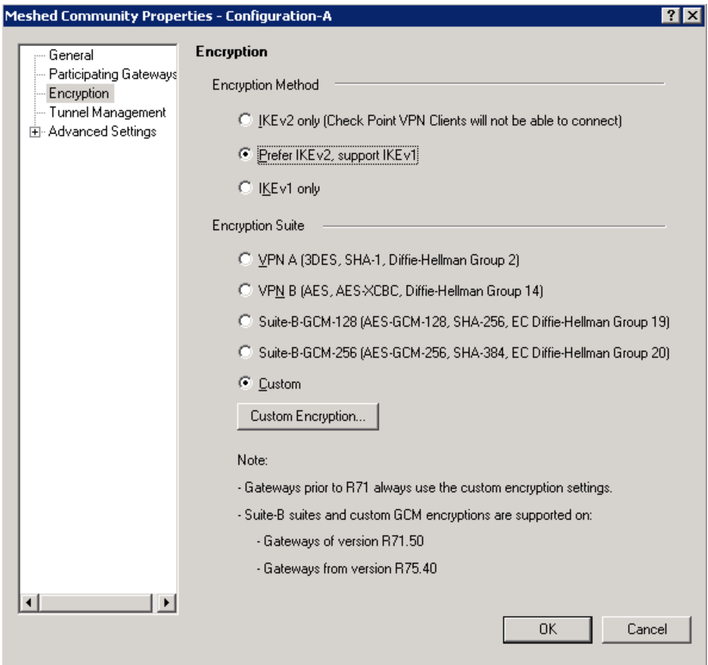


Fig. 3.2.: *VPN Community encryption properties*

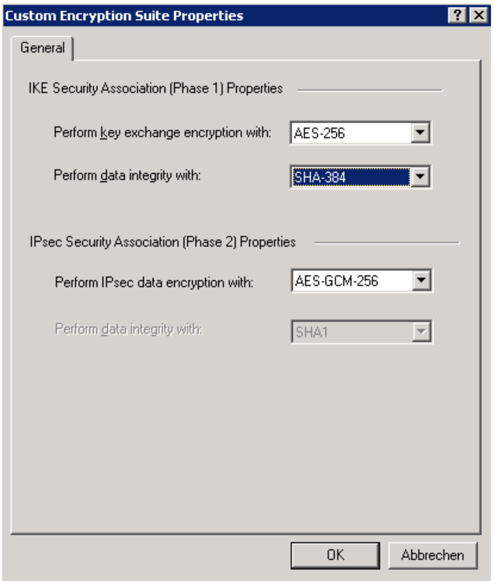


Fig. 3.3.: *Custom Encryption Suite Properties*

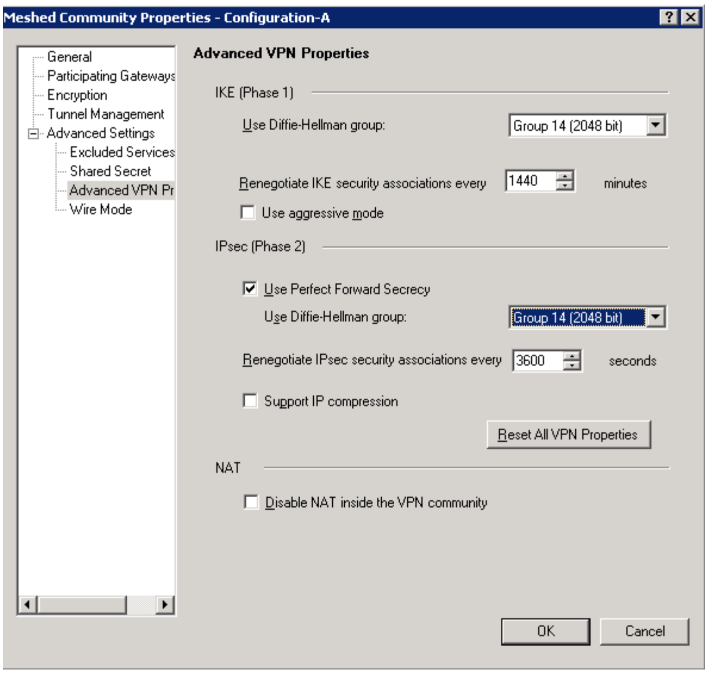


Fig. 3.4.: *Advanced VPN Properties*

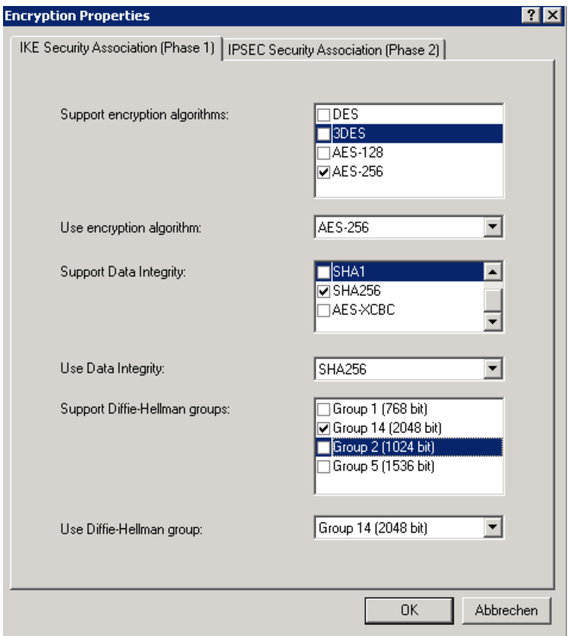


Fig. 3.5.: *Remote Access Encryption Properties*

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[13] (may require a UserCenter account to access)

3.4.3. OpenVPN

Tested with Versions

- **OpenVPN 2.3.2 from Debian "wheezy-backports" linked against openssl** (libssl.so.1.0.0)
- OpenVPN 2.2.1 from Debian Wheezy 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.

OpenVPN duplexes the tunnel into a data and a control channel. The control channel is a usual TLS connection, the data channel currently uses encrypt-then-mac CBC, see https://github.com/BetterCrypto/Applied-Crypto-Hardening/pull/91#issuecomment-75365286

Server Configuration

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

[13] https://sc1.checkpoint.com/documents/R77/CP_R77_VPN_AdminGuide/html_frameset.htm

Justification for special settings

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 section-recommendedciphers 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

- OpenVPN Documentation: Security Overview[14]

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:

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 section-keylengths for a discussion on keylengths.

In addition, edit the `pkitool` script and replace all occurrences 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*[5].

[14] https://openvpn.net/index.php/open-source/documentation/security-overview.html
[5] https://community.openvpn.net/openvpn/ticket/304

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
```

3.4.4. PPTP

PPTP is considered insecure, Microsoft recommends to "use a more secure VPN tunnel"[6].

There is a cloud service that cracks the underlying MS-CHAPv2 authentication protocol for the price of USD 200[7], and given the resulting MD4 hash, all PPTP traffic for a user can be decrypted.

3.4.5. 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 Versions

- 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
 protocol esp encryption aes-192
```

6 http://technet.microsoft.com/en-us/security/advisory/2743314
7 https://www.cloudcracker.com/blog/2012/07/29/cracking-ms-chap-v2/

```
 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

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

3.4.6. Openswan

Tested with Version

- Openswan 2.6.39 (Gentoo)

Settings

Note: the available algorithms depend on your kernel configuration (when using protostack=netkey) and/or build-time options.

To list the supported algorithms

```
$ ipsec auto --status | less
```

and look for 'algorithm ESP/IKE' at the beginning.

```
aggrmode=no
# ike format: cipher-hash;dhgroup
# recommended ciphers:
# - aes
# recommended hashes:
# - sha2_256 with at least 43 byte PSK
# - sha2_512 with at least 86 byte PSK
# recommended dhgroups:
# - modp2048 = DH14
# - modp3072 = DH15
# - modp4096 = DH16
# - modp6144 = DH17
# - modp8192 = DH18
ike=aes-sha2_256;modp2048
type=tunnel
phase2=esp
# esp format: cipher-hash;dhgroup
# recommended ciphers configuration A:
# - aes_gcm_c-256 = AES_GCM_16
# - aes_ctr-256
# - aes_ccm_c-256 = AES_CCM_16
# - aes-256
```

```
# additional ciphers configuration B:
# - camellia-256
# - aes-128
# - camellia-128
# recommended hashes configuration A:
# - sha2-256
# - sha2-384
# - sha2-512
# - null (only with GCM/CCM ciphers)
# additional hashes configuration B:
# - sha1
# recommended dhgroups: same as above
phase2alg=aes_gcm_c-256-sha2_256;modp2048
salifetime=8h
pfs=yes
auto=ignore
```

How to test

Start the vpn and using

```
$ ipsec auto --status | less
```

and look for ’IKE algorithms wanted/found’ and ’ESP algorithms wanted/loaded’.

References

- https://www.openswan.org/

3.4.7. tinc

Tested with Version

- tinc 1.0.23 from Gentoo linked against OpenSSL 1.0.1e
- tinc 1.0.23 from Sabayon linked against OpenSSL 1.0.1e

Defaults tinc uses 2048 bit RSA keys, Blowfish-CBC, and SHA1 as default settings and suggests the usage of CBC mode ciphers. Any key length up to 8196 is supported and it does not need to be a power of two. OpenSSL Ciphers and Digests are supported by tinc.

Settings Generate keys with

```
tincd -n NETNAME -K8196
```

Old keys will not be deleted (but disabled), you have to delete them manually. Add the following lines to your tinc.conf on all machines

References

- tincd(8) man page
- tinc.conf(5) man page
- tinc mailinglist: http://www.tinc-vpn.org/pipermail/tinc/2014-January/003538.html[15]

3.5. PGP/GPG - Pretty Good Privacy

The OpenPGP protocol[1] 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[2] 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 ref{section:keylengths})
- randomness (see section ref{section:RNGs})
- preference of symmetric encryption algorithm (see section ref{section:CipherSuites})
- preference of hash function (see section ref{section:CipherSuites})

Properly dealing with key material, passphrases and the web-of-trust is outside of the scope of this document. The GnuPG website[3] has a good tutorial on PGP.

This Debian How-to[194] is a great resource on upgrading your old PGP key as well as on safe default settings. This section is built based on the Debian How-to.

3.5.1. Hashing

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

[15] http://www.tinc-vpn.org/pipermail/tinc/2014-January/003538.html
[1] https://tools.ietf.org/search/rfc4880
[2] https://www.schneier.com/blog/archives/2005/02/sha1_broken.html
[3] http://www.gnupg.org/
[19] https://www.debian-administration.org/users/dkg/weblog/48
[4] https://www.debian-administration.org/users/dkg/weblog/48

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

3.6. 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.

3.7. Instant Messaging Systems

3.7.1. General server configuration recommendations

For servers, we mostly recommend to apply what's proposed by the *Peter's manifesto*[1].

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.

3.7.2. ejabberd

Tested with Versions

- ejabberd 14.12, Debian 7 Wheezy
- ejabberd 14.12, Ubuntu 14.04 Trusty
- ejabberd 15.03, Ubuntu 14.04 Trusty
- ejabberd 16.01, Ubuntu 14.04 Trusty

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

Settings

ejabberd is one of the popular Jabber servers. In order to be compliant with the manifesto, you should adapt your configuration[2]:

Additional settings

It is possible to explicitly specify a cipher string for TLS connections.

Note that we are setting the SSL option `cipher_server_preference`. This enforces our cipher order when negotiating which ciphers are used, as the cipher order of some clients chooses weak ciphers over stronger ciphers.[3]

Starting with version 15.03[4], it is possible to use custom Diffie-Hellman-Parameters. This allows us to negotiate stronger Diffie-Hellman-keys, and also helps us avoid problems with using common Diffie-Hellman-Parameters.[5] You can generate your own parameter file by running:

```
openssl dhparam -out dhparams.pem 4096
```

By default, ejabberd provides an administration website (look for the ejabberd_http module). Enable TLS protection for it like this:

Tested with Versions

- Debian Wheezy 2.1.10-4+deb7u1

Settings

Older versions of ejabberd use a different configuration file syntax. In order to be compliant with the manifesto, you should adapt your configuration[6] as follows:

Additional settings

Older versions of ejabberd (< 2.0.0) need to be patched[7] to be able to parse all of the certificates in the CA chain. Specifying a custom cipher string is only possible starting with version 13.12 (see configuration for version 14.12 above).

[2] http://www.process-one.net/docs/ejabberd/guide_en.html
[3] https://blog.thijsalkema.de/me/blog//blog/2013/09/02/the-state-of-tls-on-xmpp-3/
[4] Early versions seem to have a few bugs - although officially supported, it did not work in tests with version 15.06. Version 16.01 is confirmed to work.
[5] https://weakdh.org
[6] http://www.process-one.net/docs/ejabberd/guide_en.html
[7] http://hyperstruct.net/2007/06/20/installing-the-startcom-ssl-certificate-in-ejabberd/

References

- The ejabberd documentation: http://www.process-one.net/en/ejabberd/docs/[20]

How to test

- https://xmpp.net is a useful website to test Jabber server configurations.

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

The OTR protocol works on top of the Jabber protocol[8]. 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.

3.7.4. Charybdis

There are numerous implementations of IRC servers. In this section, we choose *Charybdis* which serves as basis for *ircd-seven*[9], developed and used by freenode. Freenode is actually the biggest IRC network[10]. *Charybdis* is part of the *Debian* & *Ubuntu* distributions.

3.7.5. SILC

SILC[11] 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.

[20] http://www.process-one.net/en/ejabberd/docs/
[8] https://otr.cypherpunks.ca/Protocol-v3-4.1.1.html
[9] https://dev.freenode.net/redmine/projects/ircd-seven
[10] http://irc.netsplit.de/networks/top10.php
[11] http://www.silcnet.org/ and https://en.wikipedia.org/wiki/SILC_(protocol)

The protocol is not really popular anymore.

3.8. Database Systems

3.8.1. Oracle

Tested with Versions

We do not test this here, since we only reference other papers for Oracle so far.

References

Technical safety requirements by *Deutsche Telekom AG* (German). Please read section 17.12 or pages 129 and following (Req 396 and Req 397) about SSL and ciphersuites http://www.telekom.com/static/-/155996/7/technische-sicherheitsanforderungen-si

3.8.2. MySQL

Tested with Versions

Debian Wheezy and MySQL 5.5

Settings

References

MySQL Documentation on SSL Connections. 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%';
```

3.8.3. DB2

Tested with Version

We do not test this here, since we only reference other papers for DB2 so far.

Settings

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
```

References

- IBM Db2 Documentation on *Supported cipher suites*. http://pic.dhe.ibm.com/infocenter/db2luw/v9r7/index.jsp?topic=%2Fcom.ibm.db2.luw.admin.sec.doc%2Fdoc%2Fc0053544.html

3.8.4. PostgreSQL

Tested with Versions

- Debian Wheezy and PostgreSQL 9.1
- Linux Mint 14 nadia / Ubuntu 12.10 quantal with PostgreSQL 9.1+136 and OpenSSL 1.0.1c

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 manually.

References

- It's recommended to read "Security and Authentication" in the manual[1].
- PostgreSQL Documentation on *Secure TCP/IP Connections with SSL*: http://www.postgresql.org/docs/9.1/static/ssl-tcp.html
- PostgreSQL Documentation on *host-based authentication*: http://www.postgresql.org/docs/current/static/auth-pg-hba-conf.html

How to test

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

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

3.9. 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.

[1] http://www.postgresql.org/docs/9.1/interactive/runtime-config-connection.html

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!

3.9.1. Bluecoat

Tested with Versions

- 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)

3.9.2. HAProxy

HAProxy can be used as loadbalancer and proxy for TCP and HTTP-based applications. Since version 1.5 it supports SSL and IPv6.

Tested with Versions

- HAProxy 1.5.11 with OpenSSL 1.0.1e on Debian Wheezy

Settings

Additional Settings

Enable NPN Support:

```
bind *:443 ssl crt server.pem npn "http/1.1,http/1.0"
```

Append the npn command in the frontend configuration of HAProxy.

Enable OCSP stapling: HAProxy supports since version 1.5.0 OCSP stapling. To enable it you have to generate the OCSP singing file in the same folder, with the same name as your certificate file plus the extension .ocsp. (e.g. your certificate file is named server.crt then the OCSP file have to be named server.crt.oscp)

To generate the OCSP file use these commands:

```
openssl x509 -in your.certificate.crt -noout -ocsp_uri # <- get your ocsp uri
openssl ocsp -noverify -issuer ca.root.cert.crt -cert your.certificate.crt -url "YOUR OCSP
↪URI" -respout your.certificate.crt.ocsp
```

Reload HAProxy and now OCSP stapling should be enabled.

Note: This OCSP signature file is only valid for a limited time. The simplest way of updating this file is by using cron.daily or something similar.

Enable HPKP: Get certificate informations:

```
openssl x509 -in server.crt -pubkey -noout | openssl rsa -pubin -outform der | openssl dgst
↪-sha256 -binary | base64
```

Then you append the returned string in the HAProxy configuration. Add the following line to the backend configuration:

```
rspadd Public-Key-Pins:\ pin-sha256="YOUR_KEY";\ max-age=15768000;\ includeSubDomains
```

Reload HAProxy and HPKP should now be enabled.

Note: Keep in mind to generate a backup key in case of problems with your primary key file.

How to test

See appendix cha-tools

3.9.3. Pound

Tested with Versions

- Pound 2.6

Settings

3.9.4. stunnel

Tested with Versions

- stunnel 4.53-1.1ubuntu1 on Ubuntu 14.04 Trusty with OpenSSL 1.0.1f, without disabling Secure Client-Initiated Renegotiation
- stunnel 5.02-1 on Ubuntu 14.04 Trusty with OpenSSL 1.0.1f
- stunnel 4.53-1.1 on Debian Wheezy with OpenSSL 1.0.1e, without disabling Secure Client-Initiated Renegotiation

Settings

Additional information

Secure Client-Initiated Renegotiation can only be disabled for stunnel versions >= 4.54, when the renegotiation parameter has been added (See changelog).

References

- stunnel documentation[21]
- stunnel changelog[22]

How to test

See appendix cha-tools

3.10. Kerberos

This section discusses various implementations of the Kerberos 5 authentication protocol on Unix and Unix-like systems as well as on Microsoft Windows.

3.10.1. Overview

Kerberos provides mutual authentication of two communicating parties, e.g. a user using a network service. The authentication process is mediated by a trusted third party, the Kerberos key distribution centre (KDC). Kerberos implements secure single-sign-on across a large number of network protocols and operating systems. Optionally, Kerberos can be used to create encrypted communications channels between the user and service.

Recommended reading

An understanding of the Kerberos protocol is necessary for properly implementing a Kerberos setup. Also, in the following section some knowledge about the inner workings of Kerberos is assumed. Therefore we strongly recommend reading this excellent introduction first: http://gost.isi.edu/publications/kerberos-neuman-tso.html. No further overview over Kerberos terminology and functions will be provided, for a discussion and a selection of relevant papers refer to http://web.mit.edu/kerberos/papers.html.

The Kerberos protocol over time has been extended with a variety of extensions and Kerberos implementations provide additional services in addition to the aforementioned KDC. All discussed implementations provide support for trust relations between multiple realms, an administrative network service (kerberos-adm, kadmind) as well as a password changing service (kpasswd). Sometimes, alternative database backends for ticket storage, X.509 and SmartCard authentication are provided. Of those, only administrative and password changing services will be discussed.

Only the Kerberos 5 protocol and implementation will be discussed. Kerberos 4 is obsolete, insecure and its use is strongly discouraged.

[21] https://www.stunnel.org/static/stunnel.html
[22] https://www.stunnel.org/sdf_ChangeLog.html

Providing a suitable Setup for secure Kerberos Operations

The aim of Kerberos is to unify authentication across a wide range of services, for many different users and use cases and on many computer platforms. The resulting complexity and attack surface make it necessary to carefully plan and continuously evaluate the security of the overall ecosystem in which Kerberos is deployed. Several assumptions are made on which the security of a Kerberos infrastructure relies:

- Every KDC in a realm needs to be trustworthy. The KDC's principal database must not become known to or changed by an attacker. The contents of the principal database enables the attacker to impersonate any user or service in the realm.
- Synchronisation between KDCs must be secure, reliable and frequent. An attacker that is able to intercept or influence synchronisation messages obtains or influences parts of the principal database, enabling impersonation of affected principals. Unreliable or infrequent synchronisation enlarges the window of vulnerability after disabling principals or changing passwords that have been compromised or lost.
- KDCs must be available. An attacker is able to inhibit authentication for services and users by cutting off their access to a KDC.
- Users' passwords must be secure. Since Kerberos is a single-sign-on mechanism, a single password may enable an attacker to access a large number of services.
- Service keytabs need to be secured against unauthorized access similarly to SSL/TLS server certificates. Obtaining a service keytab enables an attacker to impersonate a service.
- DNS infrastructure must be secure and reliable. Hosts that provide services need consistent forward and reverse DNS entries. The identity of a service is tied to its DNS name, similarly the realm a client belongs to as well as the KDC, kpasswd and kerberos-adm servers may be specified in DNS TXT and SRV records. Spoofed DNS entries will cause denial-of-service situations and might endanger:cite:MITKrb-Doc:realm_config,IETF:cat-krb-dns-locate-02 the security of a Kerberos realm.
- Clients and servers in Kerberos realms need to have synchronized clocks. Tickets in Kerberos are created with a limited, strictly enforced lifetime. This limits an attacker's window of opportunity for various attacks such as the decryption of tickets in sniffed network traffic or the use of tickets read from a client computer's memory. Kerberos will refuse tickets with old timestamps or timestamps in the future. This would enable an attacker with access to a systems clock to deny access to a service or all users logging in from a specific host.

Therefore we suggest:

- Secure all KDCs at least as strongly as the most secure service in the realm.
- Dedicate physical (i.e. non-VM) machines to be KDCs. Do not run any services on those machines beyond the necessary KDC, kerberos-adm, kpasswd and kprop services.
- Restrict physical and administrative access to the KDCs as severely as possible. E.g. ssh access should be limited to responsible adminstrators and trusted networks.
- Encrypt and secure the KDCs backups.
- Replicate your primary KDC to at least one secondary KDC.

- Prefer easy-to-secure replication (propagation in Kerberos terms) methods.Especially avoid LDAP replication and database backends. LDAP enlarges the attack surface of your KDC and facilitates unauthorized access to the principal database e.g. by ACL misconfiguration.
- Use DNSSEC. If that is not possible, at least ensure that all servers and clients in a realm use a trustworthy DNS server contacted via secure network links.
- Use NTP on a trustworthy server via secure network links.
- Avoid services that require the user to enter a password which is then checked against Kerberos. Prefer services that are able to use authentication via service tickets, usually not requiring the user to enter a password except for the initial computer login to obtain a ticket-granting-ticket (TGT). This limits the ability of attackers to spy out passwords through compromised services.

3.10.2. Implementations

Cryptographic Algorithms in Kerberos Implementations

The encryption algorithms (commonly abbreviated 'etypes' or 'enctypes') in Kerberos exchanges are subject to negotiation between both sides of an exchange. Similarly, a ticket granting ticket (TGT), which is usually obtained on initial login, can only be issued if the principal contains a version of the password encrypted with an etype that is available both on the KDC and on the client where the login happens. Therefore, to ensure interoperability among components using different implementations as shown in table table-Kerberos_enctypes, a selection of available etypes is necessary. However, the negotiation process may be subject to downgrade attacks:cite:AttKerbDepl and weak hashing algorithms endanger integrity protection and password security. This means that the des3-cbc-sha1-kd or rc4-hmac algorithms should not be used, except if there is a concrete and unavoidable need to do so. Other des3-*, des-* and rc4-hmac-exp algorithms should never be used.

Along the lines of cipher string B, the following etypes are recommended: aes256-cts-hmac-sha1-96 camellia256-cts-cmac aes128-cts-hmac-sha1-96 camellia128-cts-cmac.

Table 3.6.: *Commonly supported Kerberos encryption types by implementation. Algorithm names according to **RFC 3961**[23], except where aliases can be used or the algorithm is named differently altogether as stated~cite{,krb519,JavaJGSS,ShishiEnctypes}. See also **RFC 3962**[24], **RFC 6803**[25], **RFC 3961**[26], **RFC 4120**[27], **RFC 4120**[28].*

ID	Algorithm	MIT	Heimdal	GNU Shishi	MS ActiveDirectory
1	des-cbc-crc	✓	✓	✓	✓
2	des-cbc-md4	✓	✓	✓	
3	des-cbc-md5	✓	✓	✓	✓
6	des3-cbc-none		✓	✓	
7	des3-cbc-sha1		✓[29]		
16	des3-cbc-sha1-kd	✓[30]	✓[31]	✓	
17	aes128-cts-hmac-sha1-96	✓	✓	✓	✓[32]
18	aes256-cts-hmac-sha1-96	✓	✓	✓	✓[33]
23	rc4-hmac	✓	✓	✓	✓
24	rc4-hmac-exp	✓		✓	✓
25	camellia128-cts-cmac	✓[34]			
26	camellia256-cts-cmac	✓[34]			

Existing installations

The configuration samples below assume new installations without preexisting principals.

For existing installations:

- Existing setups should be migrated to a new master key if the current master key is using a weak enctype.
- When changing the list of supported_enctypes, principals where all enctypes are no longer supported will cease to work.
- Be aware that Kerberos 4 is obsolete and should not be used.
- Principals with weak enctypes pose an increased risk for password bruteforce attacks if an attacker gains access to the database.

To get rid of principals with unsupported or weak enctypes, a password change is usually the easiest way. Service principals can simply be recreated.

[23] https://tools.ietf.org/html/rfc3961.html
[24] https://tools.ietf.org/html/rfc3962.html
[25] https://tools.ietf.org/html/rfc6803.html
[26] https://tools.ietf.org/html/rfc3961.html
[27] https://tools.ietf.org/html/rfc4120.html
[28] https://tools.ietf.org/html/rfc4120.html
[29] named old-des3-cbc-sha1.
[30] alias des3-cbc-sha1, des3-hmac-sha1.
[31] named des3-cbc-sha1.
[32] since Vista, Server 2008.
[33] since 7, Server 2008R2.
[34] since 1.9.

MIT krb5

KDC configuration

In /etc/krb5kdc/kdc.conf set the following in your realm's configuration:

In /etc/krb5.conf set in the [libdefaults] section:

Upgrading a MIT krb5 database to a new enctype

To check if an upgrade is necessary, execute the following on the KDC in question:

```
root@kdc.example.com:~# kdb5_util list_mkeys
Master keys for Principal: K/M@EXAMPLE.COM
KVNO: 1, Enctype: des-cbc-crc, Active on: Thu Jan 01 00:00:00 UTC 1970 *
```

In this case, an old unsafe enctype is in use as indicated by the star following the key line. To upgrade, proceed as follows. First create a new master key for the database with the appropriate enctype. You will be prompted for a master password that can later be used to decrypt the database. A stash-file containing this encryption key will also be written.

```
root@kdc.example.com:~# kdb5_util add_mkey -s -e aes256-cts-hmac-sha1-96
Creating new master key for master key principal 'K/M@EXAMPLE.COM'
You will be prompted for a new database Master Password.
It is important that you NOT FORGET this password.
Enter KDC database master key:
Re-enter KDC database master key to verify:
```

Verify that the new master key has been successfully created. Note the key version number (KVNO) of the new master key, in this case 2.

```
root@kdc.example.com:~# kdb5_util list_mkeys
Master keys for Principal: K/M@EXAMPLE.COM
KVNO: 2, Enctype: aes256-cts-hmac-sha1-96, No activate time set
KVNO: 1, Enctype: des-cbc-crc, Active on: Thu Jan 01 00:00:00 UTC 1970 *
```

Set the new master key as the active master key by giving its KVNO. The active master key will be indicated by an asterisk in the master key list.

```
root@kdc.example.com:~# kdb5_util use_mkey 2
root@kdc.example.com:~# kdb5_util list_mkeys
Master keys for Principal: K/M@EXAMPLE.COM
KVNO: 2, Enctype: aes256-cts-hmac-sha1-96, Active on: Wed May 13 14:14:18 UTC 2015 *
KVNO: 1, Enctype: des-cbc-crc, Active on: Thu Jan 01 00:00:00 UTC 1970
```

Reencrypt all principals to the new master key.

```
root@kdc.example.com:~# kdb5_util update_princ_encryption
Re-encrypt all keys not using master key vno 2?
```

```
(type 'yes' to confirm)? yes
504 principals processed: 504 updated, 0 already current
```

After verifying that everything still works as desired it is possible to remove unused master keys.

```
root@kdc.example.com:~# kdb5_util purge_mkeys
Will purge all unused master keys stored in the 'K/M@EXAMPLE.COM' principal, are you sure?
(type 'yes' to confirm)? yes
OK, purging unused master keys from 'K/M@EXAMPLE.COM'...
Purging the following master key(s) from K/M@EXAMPLE.COM:
KVNO: 1
1 key(s) purged.
```

4. Theory

4.1. 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 *Practical recommendations* recommended *cipher string B*.

We start off by explaining the structure of cipher strings in section subsection-architecture (architecture) and define PFS in subsection-PFS. Next we present *Cipher String A* and *Cipher String B* in section section-recommendedciphers. 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 section-relatedPublications, the ENISA *[14]*, ECRYPT 2 *[3]* and BSI *[10]* 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 section-RNGs), keylengths (section section-keylengths), current issues in ECC (section section-EllipticCurveCryptography), a note of warning on SHA-1 (section *A note on SHA-1*) and some comments on Diffie Hellman key exchanges (section section-DH). 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 subsection-compatibility. 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 section-PKIs), 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.

4.2. Cipher suites

4.2.1. 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." *[12]*

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

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

4.2.2. 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].

[1] https://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

4.2.3. 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
```

Table 4.1.: *Configuration A ciphers*

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

4.2.4. Compatibility:

At the time of this writing only Win 7 and Win 8.1 crypto stack, OpenSSL ≥ 1.0.1e, Safari 6 / iOS 6.0.1 and Safari 7 / OS X 10.9 are covered by that cipher string.

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.

Note: All 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 *A note on SHA-1*

This results in the OpenSSL string:

```
EDH+CAMELLIA:EDH+aRSA:EECDH+aRSA+AESGCM:EECDH+aRSA+SHA256:EECDH:+CAMELLIA128:+AES128:+SSLv3:!
↪aNULL:!eNULL:!LOW:!3DES:!MD5:!EXP:!PSK:!DSS:!RC4:!SEED:!IDEA:!ECDSA:kEDH:CAMELLIA128-
↪SHA:AES128-SHA
```

Todo

make a column for cipher chaining mode

Table 4.2.: *Configuration B ciphers*

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

4.2.5. 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!}'

4.2.6. Explanation:

For a detailed explanation of the cipher suites chosen, please see section-ChoosingYourOwnCipherSuites. 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.

Straightforwardly, the authors wanted strong ciphers, forward secrecy[1] 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.
- 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.

4.2.7. Insecure Ciphers

Todo

write this.

---

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

4.2.8. Compatibility

4.2.9. Choosing your own cipher suites

Many of the parts in a cipher suite 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 section-disclaimer).

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 *[21]*. 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

Table 4.3.: *Key exchange algorithms*

	Key	EC	ephemeral
RSA	RSA		
DH	RSA		
EDH	RSA		✓
ECDH	both	✓	
EECDH	both	✓	✓
DSA	DSA		
ECDSA	DSA	✓	

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 pre-generated 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 section-EllipticCurveCryptography) 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 *[6]*. 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 section-PKIs.

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). All those mechanisms require special configuration.

Todo

Reference SRP

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.

 4.3. Random Number Generators

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.

4.3. Random Number Generators

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

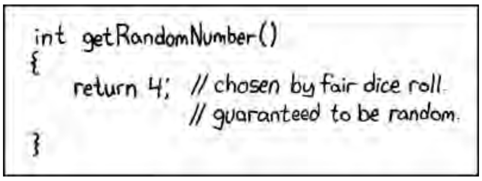


Fig. 4.1.: *xkcd, source: https://imgs.xkcd.com/comics/random_number.png, license: CC-BY-NC*

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.

4.3.1. 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 over-estimated. The most critical component that has been shown to return wrong results in an emulated environment is the timing source *[9][5]*.

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 [11].

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 [11].

4.3.2. Linux

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 [11].

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
```

4.3.3. 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 [19].

For Linux the `haveged` implementation [1] based on the HAVEGE [16] 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 [2]. 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 [11].

4.4. 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

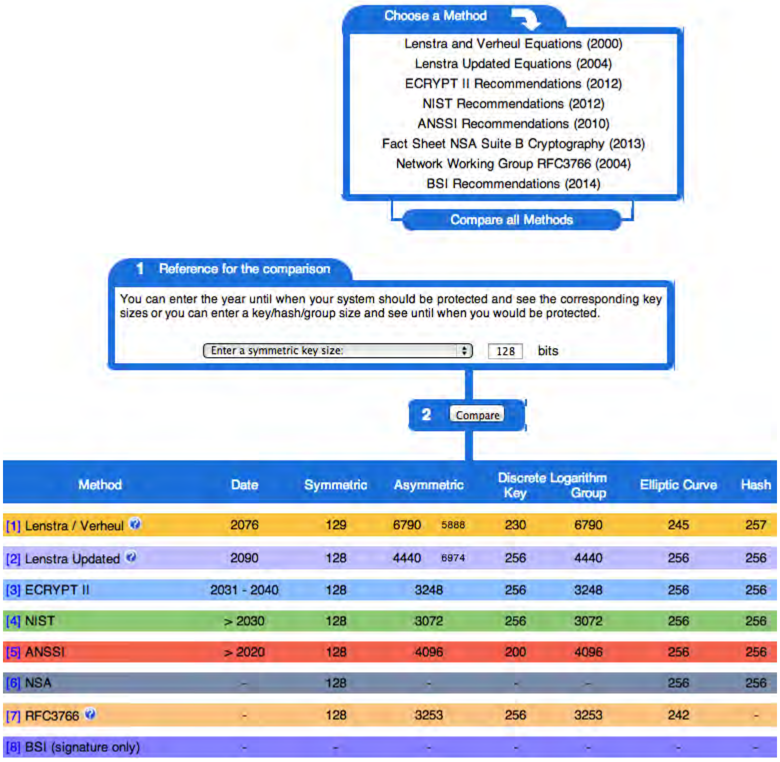
Recommendations on keylengths need to be adapted regularly. Since this document first of all is static and second of all, does not consider itself to be authoritative on keylengths, we would rather refer to existing publications and websites. Recommending a safe key length is a hit-and-miss issue.

Furthermore, when choosing an encryption algorithm and key length, 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 *[3]* 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 [**?**, 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 fig-keylengths.com shows a typical comparison of keylengths on this web site.



Method	Date	Symmetric	Asymmetric	Discrete Logarithm Key	Discrete Logarithm Group	Elliptic Curve	Hash
[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)	-	-	-	-	-	-	-

4.4.1. Summary

- For asymmetric public-key cryptography we consider any key length below 3248 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.

4.4.2. 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[1], it is clear that 3DES can only be considered to provide for 80 bits / 112 bits security.

[1] http://csrc.nist.gov/publications/PubsSPs.html#800-57-part1, pages 63 and 64

4.5. 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 *[20][13][18]*. 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[1][2][3]. ECC provides for much stronger security with less computationally expensive operations in comparison to traditional asymmetric algorithms (See the Section section-keylengths). 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)[4] which were later widely implemented in most common crypto libraries. Those parameters came under question repeatedly from cryptographers *[7][15][17]*. At the time of writing, there is ongoing research as to the security of various ECC parameters *[6]*. 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!

4.6. 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 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.

[1] http://arstechnica.com/security/2013/10/a-relatively-easy-to-understand-primer-on-elliptic-curve-cryptography
[2] https://www.imperialviolet.org/2010/12/04/ecc.html
[3] http://www.isg.rhul.ac.uk/~sdg/ecc.html
[4] http://www.nist.gov

4.7. A note on Diffie Hellman Key Exchanges

A common question is which Diffie Hellman (DH) Parameters should be used for Diffie Hellman key exchanges[1]. We follow the recommendations in ECRYPT II [**?**, chapter 16]

Where configurable, we recommend using the Diffie Hellman groups defined for IKE, specifically groups 14-18 (2048–8192 bit MODP **RFC 3526**[35]). These groups have been checked by many eyes and can be assumed to be secure.

For convenience, we provide these parameters as PEM files on our webserver[2].

4.8. 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 sec-hardeningpki and *[8]*.

The Web of Trust is a decentralized system where people sign each other's 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.

4.8.1. 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.

[1] http://crypto.stackexchange.com/questions/1963/how-large-should-a-diffie-hellman-p-be
[35] https://tools.ietf.org/html/rfc3526.html
[2] https://www.bettercrypto.org/static/dhparams/

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> -sha256
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
```

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, you can self-sign a previously created certificate with this command:

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

You can also create a self-signed certificate in just one command:

```
openssl req -new -x509 -keyout privkey.pem -out cacert.pem -days 1095 -nodes -newkey rsa:
↪<keysize> -sha256
```

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.

4.8.2. 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 *[28]*. Recently Google found certificates issued to them, which were not used by the company *[27]*. 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 *[25]*.

Therefore several security enhancements were introduced by different organizations and vendors *[29]*. Currently two methods are used, DANE **RFC 6698**[36] and Certificate Pinning *[26]*. Google recently proposed a new system to detect malicious CAs and certificates called Certificate Transparency *[24]*. In addition, **RFC 6844**[37] describes Certification Authorization Records, a mechanism for domain name owners to signal which Certificate Authorities are authorized to issue certificates for their domain.

4.8.3. Certification Authorization Records

RFC 6844[38] describes Certification Authorization Records, a mechanism for domain name owners to signal which Certificate Authorities are authorized to issue certificates for their domain.

[36] https://tools.ietf.org/html/rfc6698.html
[37] https://tools.ietf.org/html/rfc6844.html
[38] https://tools.ietf.org/html/rfc6844.html

When a CAA record is defined for a particular domain, it specifies that the domain owner requests Certificate Authorities to validate any request against the CAA record. If the certificate issuer is not listed in the CAA record, it should not issue the certificate.

The RFC also permits Certificate Evaluators to test an issued certificate against the CAA record, but should exercise caution, as the CAA record may change during the lifetime of a certificate, without affecting its validity.

CAA also supports an iodef property type which can be requested by a Certificate Authority to report certificate issue requests which are inconsistent with the issuer's Certificate Policy.

Configuration of CAA records

BIND supports CAA records as of version 9.9.6.

A CAA record can be configured by adding it to the zone file:

```
$ORIGIN example.com
       CAA 0 issue "ca1.example"
       CAA 0 iodef "mailto:security@example.com"
```

If your organization uses multiple CA's, you can configure multiple records:

```
CAA 0 issue "ca1.example"
CAA 0 issue "ca2.example"
```

"ca1.example" and "ca2.example" are unique identifiers for the CA you plan on using. These strings can be obtained from your Certificate Authority, and typically are its top level domain. An example is "letsencrypt.org" for the Let's Encrypt CA operated by the Internet Security Research Group.

Knot-DNS supports CAA records as of version 2.2.0.

Validation of CAA records

Once a CAA record is deployed, it can be validated using the following dig query:

```
user@system:~$ dig CAA google.com

; <<>> DiG 9.10.3-P4-Debian <<>> CAA google.com

;; ANSWER SECTION:
google.com.          3600 IN   CAA  0 issue "symantec.com"
```

On older versions of Dig, which do not support CAA records, you can query the record type manually:

```
dig +short -t TYPE257 google.com
\# 19 0005697373756573796D616E7465632E636F6D
```

4.9. TLS and its support mechanisms

4.9.1. HTTP Strict Transport Security (HSTS)

HTTP Strict Transport Security (HSTS) is a web security policy mechanism. HSTS is realized through HTTP header by which a web server declares that complying user agents (web browsers) should interact with it by using *only* secure HTTPS connections[1].

HSTS header is bound to a DNS name or domain by which the server was accessed. For example if server serves content for two domains and it is HTTPS enabled only for one domain, the browser won't enforce HSTS for the latter.

HSTS reduces the risk of active man-in-the-middle attacks such as SSL stripping, and impersonation attacks with *untrusted* certificate. HSTS also helps to avoid unintentional mistakes such as insecure links to a secure web site (missing HTTPS links[2]), and mistyped HTTPS URLs.

After the web browser receives a HSTS header in a *correctly*[3] prepared SSL session it will automatically use secure HTTPS links for accessing the server. This prevents unencrypted HTTP access (SSL striping, mistyped HTTPS URLs, etc.) when the server is accessed later by the client.

When a server (that previously emitted a HSTS header) starts using untrusted certificate, complying user agent must show an error message and *block the server connection*. Thus impersonation MITM attack with *untrusted* certificate cannot occur.

For the initial setup HSTS header needs a trusted secure connection over HTTPS. This limitation can be addressed by compiling a list of STS enabled sites directly into a browser[4].

HSTS Header Directives

HSTS header can be parametrized by two directives:

- max-age=<number-of-seconds>
- includeSubdomains

max-age is a required directive. This directive indicates the number of seconds during which the user agent should enforce the HSTS policy (after the reception of the STS header field from a server).

includeSubdomains is an optional directive. This directive indicates that the HSTS Policy applies to this HSTS Host as well as *any subdomains of the host's domain name*.

[1] https://en.wikipedia.org/wiki/HTTP_Strict_Transport_Security
[2] Thus, it might be useful for fixing HTTPS mixed-content related errors, see https://community.qualys.com/blogs/securitylabs/2014/03/19/https-mixed-content-still-the-easiest-way-to-break-ssl.
[3] Website must load without SSL/TLS browser warnings (certificate is issued by a trusted CA, contains correct DNS name, it is time valid, etc.)
[4] List of the preloaded sites can be found at http://dev.chromium.org/sts. This list is managed by Google/Chrome but it is also used by Firefox https://wiki.mozilla.org/Privacy/Features/HSTS_Preload_List

HSTS Client Support

HSTS is supported[5] by these web browsers:

- Firefox version >= v4.0
- Chrome version >= 4.0
- Android Browser >=4.4
- Opera version >= 12.0
- Opera mobile >= 16.0
- Safari >= 7.0
- Microsoft Internet Explorer >= 11 (with update provided 09. June 2015)
- Microsoft Edge >= 12

HSTS Considerations

Before enabling HSTS it is recommended to consider following:

- Is it *required* to serve content or services over HTTP?
- Enabling *includeSubdomains* and SSL certificate management.
- Proper value of *max-age*.

It is recommended to serve all content using HTTPS, but there are exceptions to this rule as well. Consider running a private PKI[6]. CRLs and OCSP responses are published typically by HTTP protocol. If HSTS is enabled on the site where OCSP and CRLs are published the browser might fail fetching CRL or validating OCSP response.

Similar reasoning goes for *includeSubdomains*. One needs to be sure that HTTPS can be enforced for all subdomains. Moreover the administrators are advised to watch for expiration of the SSL certificate and handle the renewal process with caution. If a SSL certificate is renewed after expiration or misses a (HSTS enabled) domain name, the connection to site will break (without providing override mechanism to the end user).

Finally HSTS should be tested with lower *max-age* values and deployed with higher *max-age* values.

Testing HSTS

HSTS can be tested either using locally or through the Internet.

[5] http://caniuse.com/stricttransportsecurity
[6] see **??**

For local testing it is possible to utilize Chrome Web browser UI by typing chrome://net-internals/#hsts[7] in the address bar.

Testing over the Internet can be conducted by Qualys SSL Labs test https://www.ssllabs.com/ssltest/. *Strict Transport Security (HSTS)* information is located in the *Protocol Details* section.

References

- Websites Must Use HSTS in Order to Be Secure https://www.eff.org/deeplinks/2014/02/websites-hsts
- OWASP: HTTP Strict Transport Security: https://www.owasp.org/index.php/HTTP_Strict_Transport_Security
- HSTS Browser Compatibility List: http://caniuse.com/stricttransportsecurity
- RFC 6797:HTTP Strict Transport Security (HSTS) - Examples: https://tools.ietf.org/html/rfc6797#section-6.2

4.9.2. HTTP Public Key Pinning (HPKP)

Much like HTTP Strict Transport Security (HSTS), HTTP Public Key Pinning (HPKP) is a Trust On First Use (TOFU) mechanism. It protects HTTPS websites from impersonation using certificates issued by compromised certificate authorities. The data for Pinning is supplied by an HTTP-Header sent by the WebServer.

HPKP Header Directives

HPKP provides two different types of headers:

- Public-Key-Pins
- Public-Key-Pins-Report-Only

HPKP header can be parametrized by following directives:

- pin-sha256="<YOUR_PUBLICKEY_HASH=>"
- max-age=<number-of-seconds>
- includeSubdomains
- report-uri="<https://YOUR.URL/TO-REPORT>"

pin-sha256 is a required directive. It can and should be used several (at least two) times for specifying the public keys of your domain-certificates or CA-certificates. Operators can pin any one or more of the public keys

[7] see http://blog.chromium.org/2011/06/new-chromium-security-features-june.html

in the certificate-chain, and indeed must pin to issuers not in the chain (as, for example, a Backup Pin). Pinning to an intermediate issuer, or even to a trust anchor or root, still significantly reduces the number of issuers who can issue end-entity certificates for the Known Pinned Host, while still giving that host flexibility to change keys without a disruption of service. OpenSSL can be used to convert the Public Key of an X509-Certificate as follows:

```
openssl x509 -in <certificate.cer> -pubkey -noout |
 openssl rsa -pubin -outform der |
 openssl dgst -sha256 -binary |
 openssl enc -base64
writing RSA key
pG3WsstDsfMkRdF3hBClXRKYxxKUJIOu8DwabG8MFrU=
```

This piped usage of OpenSSL first gets the Public-Key of <certificate.cer>, converts it do DER (binary) format, calculates an SHA256 Hash and finally encodes it Base64. The output (including the ending Equal-Sign) is exactly whats needed for the *pin-sha256="<YOUR_PUBLICKEY_HASH=>"* parameter. To generate the Hash for a prepared Backup-Key just create a Certificate-Signing-Request and replace "`openssl x509`" by "`openssl req -in <backup-cert.csr> -pubkey -noout`" as first OpenSSL command.
Instead of using OpenSSL even WebServices like https://report-uri.io/home/pkp_hash/ can be used to get a suggestion for the possible Public-Key-Hashes for a given WebSite.

max-age is a required directive (when using the *Public-Key-Pins* header). This directive specifies the number of seconds during which the user agent should regard the host (from whom the message was received) as a Known Pinned Host.

includeSubdomains is an optional directive. This directive indicates that the same pinning applies to this Host as well as *any subdomains of the host's domain name*. Be careful - you need to use a Multi-Domain/Wildcard-Certificate or use the same Pub/Private-Keypair in all Subdomain-Certificates or need to pin to CA-Certificates signing all your Subdomain-Certificates.

report-uri is an optional directive. The presence of a report-uri directive indicates to the Browser that in the event of Pin Validation failure it should post a report to the report-uri (HTTP-Post is done using JSON, Syntax see RFC-7469 Section 3[1]). There are WebServices like https://report-uri.io/ out there which can be used to easily collect and visualize these reports.

HPKP Client Support

HPKP is supported[2] by these web browsers:

- Firefox version >= 35
- Chrome version >= 38

[1] https://tools.ietf.org/html/rfc7469#section-3
[2] http://caniuse.com/#feat=publickeypinning

- Android Browser >= 44
- Opera version >= 25

Currently (16. Oct 2015) there is no HPKP Support in: Apple Safari, Microsoft Internet Explorer and Edge.

HPKP Considerations

Before enabling HPKP it is recommended to consider following:

- Which Public-Keys to use for Pinning (Certificat + Backup-Certificate, CAs, Intermediate-CAs)
- Proper value of *max-age*. Start testing with a short Period, increase Period after deployment.
- Be careful when using *includeSubdomains*, are all your subdomains covered by the defined Public-Key-Hashes?

The administrators are advised to watch for expiration of the SSL certificate and handle the renewal process with caution. If a SSL certificate is renewed without keeping the Public-Key (reusing the CSR) for an HPKP enabled domain name, the connection to site will break (without providing override mechanism to the end user).

Testing HPKP

HPKP can be tested either using locally or through the Internet.

There is a handy Bash-Script which uses OpenSSL for doing several SSL/TLS-Tests available at https://testssl.sh/

```
wget -q https://testssl.sh/testssl.sh
wget -q https://testssl.sh/mapping-rfc.txt
chmod 755 ./testssl.sh
./testssl.sh https://your-domain.com

# Sample Output, just HSTS and HPKP Section (Full report is quite long!):
Strict Transport Security    182 days=15724800 s, includeSubDomains
Public Key Pinning # of keys: 2, 90 days=7776000 s, just this domain
           matching host key: pG3WsstDsfMkRdF3hBClXRKYxxKUJIOu8DwabG8MFrU
```

For local testing it is possible to utilize Google Chrome Web browser, just open the Chrome net-internals-URL: chrome://net-internals/#hsts.

For Mozilla Firefox there is an Plug-In provided by the "Secure Information Technology Center Austria" available: https://demo.a-sit.at/firefox-plugin-highlighting-safety-information/

Testing over the Internet can be conducted by Qualys SSL Labs test https://www.ssllabs.com/ssltest/. *Public Key Pinning (HPKP)* information is located in the *Protocol Details* section.

There is also a fast online HPKP-only check at https://report-uri.io/home/pkp_analyse.

References

- OWASP: Certificate and Public Key Pinning: https://www.owasp.org/index.php/Certificate_and_Public_Key_Pinning
- HPKP Browser Compatibility List: http://caniuse.com/#feat=publickeypinning
- RFC 7469:Public Key Pinning Extension for HTTP - Examples: https://tools.ietf.org/html/rfc7469#section-2.1.5

A. Tools

This section lists tools for checking the security settings.

A.1. SSL & TLS

Server checks via the web

- ssllabs.com[39] 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[40]
- Check SPDY protocol support and basic TLS setup[41]
- XMPP/Jabber Server check (Client-to-Server and Server-to-Server)[42]
- Luxsci SMTP TLS Checker[43]
- Does your mail server support StartTLS?[44]
- http://checktls.com is a tool for testing arbitrary TLS services.
- TLS and SSH key check[45]
- http://tls.secg.org is a tool for testing interoperability of HTTPS implementations for ECC cipher suites.
- http://www.whynopadlock.com/ Testing for mixed SSL parts loaded via http that can totally lever your HTTPS.

Browser checks

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

39 https://ssllabs.com
40 https://www.sslshopper.com/ssl-checker.html
41 http://spdycheck.org/
42 https://xmpp.net/
43 https://luxsci.com/extranet/tlschecker.html
44 https://starttls.info/
45 https://factorable.net/keycheck.html

- Check Browsers SSL/TLS support and vulnerability to attacks: https://www.howsmyssl.com

Command line tools

- https://sourceforge.net/projects/sslscan connects to a given SSL service and shows the cipher suites that are offered.
- http://www.bolet.org/TestSSLServer/ tests for BEAST and CRIME vulnerabilities.
- https://github.com/drwetter/testssl.sh checks a server's service on any port for the support of TLS/SSL ciphers, protocols as well as some cryptographic flaws (CRIME, BREACH, CCS, Heartbleed).
- https://github.com/iSECPartners/sslyze Fast and full-featured SSL scanner
- https://github.com/jvehent/cipherscan Fast TLS scanner (ciphers, order, protocols, key size and more)
- http://nmap.org/ nmap security scanner
- http://www.openssl.net OpenSSL s_client

A.2. Key length

- Comprehensive online resource for comparison of key lengths according to common recommendations and standards in cryptography.[46]

A.3. RNGs

- ENT[47] is a pseudo random number generator sequence tester.
- HaveGE[48] 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[49] a random number generator testing tool.
- CAcert Random[50] another random number generator testing service.

[46] http://www.keylength.com
[47] http://www.fourmilab.ch/random/
[48] http://www.issihosts.com/haveged/
[49] http://www.phy.duke.edu/~rgb/General/dieharder.php
[50] http://www.cacert.at/random/

A.4. Guides

- See https://www.ssllabs.com/downloads/SSL_TLS_Deployment_Best_Practices.pdf

B. Links

- IANA official list of Transport Layer Security (TLS) Parameters[51]
- SSL cipher settings[52]
- Elliptic curves and their implementation (04 Dec 2010)[53]
- A (relatively easy to understand) primer on elliptic curve cryptography[54]
- Duraconf, A collection of hardened configuration files for SSL/TLS services (Jacob Appelbaum's github)[55]
- Attacks on SSL a comprehensive study of BEAST, CRIME, TIME, BREACH, LUCKY 13 & RC4 Biases[56]
- EFF How to deploy HTTPS correctly[57]
- Bruce Almighty: Schneier preaches security to Linux faithful (on not recommending to use Blowfish anymore in favor of Twofish)[58]
- Implement FIPS 183-3 for DSA keys (1024bit constraint)[59]
- Elliptic Curve Cryptography in Practice[60]
- Factoring as a Service[61]
- Black Ops of TCP/IP 2012[62]
- SSL and the Future of Authenticity, Moxie Marlinspike - Black Hat USA 2011[63]
- ENISA - Algorithms, Key Sizes and Parameters Report (Oct.'13)[64]
- Diffie-Hellman Groups[65]

---

51 https://www.iana.org/assignments/tls-parameters/tls-parameters.txt
52 http://www.skytale.net/blog/archives/22-SSL-cipher-setting.html
53 https://www.imperialviolet.org/2010/12/04/ecc.html
54 http://arstechnica.com/security/2013/10/a-relatively-easy-to-understand-primer-on-elliptic-curve-cryptography
55 https://github.com/ioerror/duraconf
56 https://www.isecpartners.com/media/106031/ssl_attacks_survey.pdf
57 https://www.eff.org/https-everywhere/deploying-https
58 https://www.computerworld.com.au/article/46254/bruce_almighty_schneier_preaches_security_linux_faithful/?pp=3
59 https://bugzilla.mindrot.org/show_bug.cgi?id=1647
60 http://eprint.iacr.org/2013/734.pdf
61 http://crypto.2013.rump.cr.yp.to/981774ce07e51813fd4466612a78601b.pdf
62 http://dankaminsky.com/2012/08/06/bo2012/
63 https://www.youtube.com/watch?v=Z7Wl2FW2TcA
64 http://www.enisa.europa.eu/activities/identity-and-trust/library/deliverables/algorithms-key-sizes-and-parameters-report
65 http://ibm.co/18lslZf

- Diffie-Hellman Groups standardized in RFC3526~cite{rfc3526}[66]
- ECC-enabled GnuPG per RFC6637~cite{rfc6637}[67]
- TLS Security (Survey + Lucky13 + RC4 Attack) by Kenny Paterson[68]
- Ensuring High-Quality Randomness in Cryptographic Key Generation[69]
- Wikipedia: Ciphertext Stealing[70]
- Wikipedia: Malleability (Cryptography)[71]
- Ritter's Crypto Glossary and Dictionary of Technical Cryptography[72]

[66] https://datatracker.ietf.org/doc/rfc3526/
[67] https://code.google.com/p/gnupg-ecc
[68] https://www.cosic.esat.kuleuven.be/ecc2013/files/kenny.pdf
[69] http://arxiv.org/abs/1309.7366v1
[70] https://en.wikipedia.org/wiki/Ciphertext_stealing
[71] https://en.wikipedia.org/wiki/Malleability_(cryptography)
[72] 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. SSL libraries

Most if not all of the cryptographic work is done by the SSL libraries installed on your system. Supported protocols, cipher suites and more depend on the version of the SSL library in use. Whenever you upgrade the SSL library, a recompile of all applications using that library is required to use the newly available features. Some features not only require a SSL library supporting it, but also the application using that feature. An example for that may be Apache supporting elliptic curve cryptography only from version 2.4 onwards, no matter if OpenSSL supported it or not.

As explained above, creating a secure setup isn't just a matter of configuration but also depends on several other factors with the most important being the SSL libraries and their support of protocols and cipher suites. Furthermore, applications actually need to make use of those.

For most configuration snippets throughout this paper we used OpenSSL's cipher strings. Sadly they are different from the official IANA standard names. When you use a different library like for example GnuTLS (which is quite common on Debian systems) you might need to change the cipher string. The hex code for a cipher string however is common to all versions and and library implementations: `TLS_RSA_CAMELLIA_256_CBC_SHA1` in GnuTLS is equivalent to `CAMELLIA256-SHA` in OpenSSL and `TLS_RSA_WITH_CAMELLIA_256_CBC_SHA` in the IANA standard with the hex code `0x00,0x84` as specified in **RFC 5932**[73]. Section *Cipher Suite Name Cross-Reference* lists all currently defined cipher suites with their codes and both names.

Regardless of this clash of nomenclature, as a sysadmin you are required to check what the SSL libraries on your systems support on how you may get the most security out of your systems.

D.1. Priority strings

Choosing cipher strings requires the use of an intermediate language that allows selection and deselection of ciphers, key exchange mechanisms, MACs and combinations of those. Common combinators consist of `+`, `-` and `!`

combinator	effect	example
`+`	add at this position	`ALL:+SHA256`
`-`	remove at the current position	`ALL:-SSLv3`
`!`	permanently remove from selection	`ALL:!3DES:!RC4`
(OpenSSL) `@`	special command	`ALL:@STRENGTH`
(GnuTLS) `%`	special command	`NORMAL:%NEW_PADDING`

A list of special strings to use can be found in http://www.gnutls.org/manual/html_node/Priority-Strings.html for GnuTLS or https://www.openssl.org/docs/apps/ciphers.html for OpenSSL. There is, however, no common syntax for a cipher string throughout different SSL libraries.

[73] https://tools.ietf.org/html/rfc5932.html

E. 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 Thu Dec 1 18:25:42 2016.

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

Table E.1.: *Cipher Suites*

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
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	

Continued on next page

Table E.1 – continued from previous page

Code	IANA Name	OpenSSL Name
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
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	

Continued on next page

Table E.1 – continued from previous page

Code	IANA Name	OpenSSL Name
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
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	

Continued on next page

Table E.1 – continued from previous page

Code	IANA Name	OpenSSL Name
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
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

Continued on next page

Table E.1 – continued from previous page

Code	IANA Name	OpenSSL Name
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
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	

Continued on next page

Table E.1 – continued from previous page

Code	IANA Name	OpenSSL Name
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	
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	
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	

Continued on next page

Table E.1 – continued from previous page

Code	IANA Name	OpenSSL Name
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	
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	

F. 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.

- Lync
- 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
- LDAP
- seclayer-tcp
- Commerical network equipment vendors
- RADIUS
- Moxa , APC, und co... ICS . Ethernet to serial
- rsyslog
- v6 spoofing (look at work by Ferndo Gont, Marc Heuse, et. al.)
- tinc
- racoon
- l2tp
- telnets
- ftps
- DSL modems (where to start?)
- UPnP, natPmp
- SAML federated auth providers footnote{e.g., all the REFEDS folks (url{https://refeds.org/}), including InCommon (url{http://www.incommon.org/federation/metadata.html} url{https://wiki.shibboleth.net/confluence/display/SHIB2/TrustManagement})}
- Microsoft SQL Server
- Microsoft Exchange
- HAProxyfootnote{url{https://lists.cert.at/pipermail/ach/2014-November/001601.html}}
- HTTP Key Pinning (HTKP)
- IBM HTTP Server
- Elastic Load Balancing (ELB)footnote{url{https://lists.cert.at/pipermail/ach/2014-May/001422.html}}

F.1. Software not covered by this guide

- telnet: Usage of telnet for anything other than fun projects is highly discouraged
- Simple Network Management Protocol (SNMP): Remote Management Software should not be available from a routed network. There is an inestimable number of problems with these implementations. Popular vendors regularly have exploits or DDoS problems with their embedded remote management and are suffering from SNMP stacks.footnote{url{https://lists.cert.at/pipermail/ach/2014-May/001389.html}} Tunneling these services over SSH or stunnel with proper authentication can be used if needed.
- Puppet DB: A Proxy or a tunnel is recommended if it needs to be facing public network interfaces.footnote{url{https://lists.cert.at/pipermail/ach/2014-November/001626.html}}
- rsync: Best use it only via SSH for an optimum of security and easiest to maintain.

G. 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
- Dirksen-Thedens, Mathis
- Dulaunoy, Alexandre
- Gühring Philipp
- Grigg, Ian
- Haslinger, Gunnar
- Huebl, Axel
- Kovacic, Daniel
- Lenzhofer, Stefan
- Lorünser, Thomas
- Maass, Max
- Mehlmauer, Christian
- Millauer, Tobias
- Mirbach, Andreas
- O'Brien, Hugh
- Pacher, Christoph
- Palfrader, Peter
- Pape, Tobias (layout)
- Petukhova, Anna (Logo)
- Pichler, Patrick
- Riebesel, Nicolas
- Roeckx, Kurt
- Roesen, Jens
- Rublik, Martin
- Schüpany, Mathias
- Schwarz, René («DigNative»)
- Seidl, Eva (PDF layout)
- Van Horenbeeck, Maarten
- 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.

H. Glossary

cipher An algorithm for performing encryption or decryption—a series of well-defined steps that can be followed as a procedure.

Bibliography

[1] Haveged – a simple entropy daemon. December 2013. Accessed 2013-12-06. URL: http://www.issihosts.com/haveged/.

[2] Haveged – a simple entropy daemon: runtime testing. December 2013. Accessed 2013-12-06. URL: http://www.issihosts.com/haveged/.

[3] Nigel Smart (Ed.). Ecrypt ii yearly report on algorithms and keysizes. Technical Report, European Network of Excellence in Cryptology II, 2012. URL: http://www.ecrypt.eu.org/ecrypt2/documents/D.SPA.20.pdf.

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

[5] Jacob Appelbaum, Marsh Ray, and Oscar Koeroo. Weak random number generation within virtualized environments. Security Advisory 2011-02, PolarSSL, December 2011. URL: https://polarssl.org/tech-updates/security-advisories/polarssl-security-advisory-2011-02.

[6] Daniel J. Bernstein. Safecurves: choosing safe curves for elliptic-curve cryptography. December 2013. Accessed 2013-12-09. URL: http://safecurves.cr.yp.to/rigid.html.

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

[8] 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. URL: https://jhalderm.com/pub/papers/https-imc13.pdf.

[9] Jakob Engblom. Evaluating HAVEGE randomness. February 2011. Blog: Observations from Uppsala. URL: http://jakob.engbloms.se/archives/1374.

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

[11] 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. URL: https://factorable.net/weakkeys12.extended.pdf.

[12] J. Katz and Y. Lindell. *Introduction to modern cryptography*. Chapman and Hall/CRC Cryptography and Network Security Series. Chapman & Hall/CRC, 2008. ISBN 9781584885511. URL: http://books.google.at/books?id=WIc_AQAAIAAJ.

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

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

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

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

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

[18] Eric W. Weisstein. Elliptic curve. Math Dictionary Entry, \hyperref http://mathworld.wolfram.com/Wolfram \hyperref http://mathworld.wolfram.com/Research \hyperref http://mathworld.wolfram.com/Mathworld, December 2013. Accessed 2013-12-12. URL: http://mathworld.wolfram.com/EllipticCurve.html.

[19] Wikipedia. /dev/random — wikipedia, the free encyclopedia. 2016. [Online; accessed 1-December-2016]. URL: https://en.wikipedia.org/w/index.php?title=/dev/random&oldid=752512186.

[20] Wikipedia. Discrete logarithm — wikipedia, the free encyclopedia. 2016. [Online; accessed 21-November-2016]. URL: https://en.wikipedia.org/w/index.php?title=Discrete_logarithm&oldid=750730405.

[21] Wikipedia. Export of cryptography from the united states — wikipedia, the free encyclopedia. 2016. [Online; accessed 19-November-2016]. URL: https://en.wikipedia.org/w/index.php?title=Export_of_cryptography_from_the_United_States&oldid=750316577.

[22] Wikipedia. Tempest (codename) — wikipedia, the free encyclopedia. 2016. [Online; accessed 17-November-2016]. URL: https://en.wikipedia.org/w/index.php?title=Tempest_(codename)&oldid=750068410.

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

[24] Adam Langley, Ben Laurie, Emilia Kasper. Certificate Transparency. 07 2013. URL: http://www.certificate-transparency.org.

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

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

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

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

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

Index