Co:Z® Co-Processing Toolkit for z/OS

z/OS OpenSSH - Quick Install Guide

2.0.0 Edition

Published January 2018

Table of Contents

Revision History

Version 2.0.0 - January 8, 2018

- Revised for APAR OA54299 on IBM z/OS V2R2 OpenSSH and z/OS V2R3 OpenSSH.

1. Basic Installation and Configuration

1.1 Introduction

This guide is designed to help systems programmers quickly configure z/OS - OpenSSH. This guide assumes OpenSSH APAR OA54299 is installed on z/OS V2R2 or V2R3 or a later z/OS release. With this APAR installed, IBM z/OS OpenSSH will directly use the `CPACF` instruction, when present, to implement symmetric ciphers and MAC algorithms. This configuration is preferred over our prior recommendation to use `ICSF`.

While the procedures in this document will work in most environments, users should reference the appropriate IBM documentation as appropriate. The primary reference is the *z/OS OpenSSH User's Guide*. This guide will call out specific sections of the User's Guide or other documents for additional information.

Updated for OpenSSH running on z/OS V2R2 or V2R3 with APAR OA54299

This version of the quick install guide has been updated specifically for the the new functionality added to OpenSSH with this APAR: CPACF support. If you do not have this APAR installed, refer to *IBM Ported Tools OpenSSH v1.3 - Quick Install Guide*, which is compatible with z/OS OpenSSH V2R2 / V2R3.

Topics covered in this guide:

- Prerequisites, service planning
- Language Environment tuning considerations
- ICSF support for secure random numbers via `/dev/random`
- Configuration files, started task, etc.
- z/OS Communications Server TCP/IP, Resolver and syslogd considerations
- CPACF support for hardware accelerated ciphers and MACs
- Managing the `/tmp` filesystem

Note: The included examples assume that you are running RACF as your system security product. z/OS OpenSSH will also work with *CA-ACF2* and *CA-TSS*, but you will be required to translate RACF commands as shown to those products. If you have one of those products and would like to contribute tested examples, please contact us.

1.2 Prerequisites

This guide assumes that you are running OpenSSH on z/OS V2V2 or later. Using this product and exploiting these features requires:

- APAR OA54299: provides CPACF support on V2R2 or V2R3
- CPACF - processor feature 3863 (free and enabled by default in most countries)
- ICSF installed and running (even if you don't have a co-processor card)

1.3 Install / Service Planning

- Review and install as appropriate any service for OpenSSH (HOS2220 or HOS2230). See upgrade ZOSV2R2/3 Subset ZOSOSSH
- Be sure to install the PTF for APAR OA54299.
- Review and install as appropriate ICSF and its required service.

1.4 Check file attributes and ownership

From a z/OS Unix shell, check the permissions and owner of the following directories:

```
$ ls -ld /etc/ssh /var/empty /var/run
drwxrwxrwx❶  2 STC1  ❷  SYS1        8192 Feb 25 14:30 /etc/ssh
drwxr-xr-x   3 STC1     SYS1        8192 Feb 21  2013 /var/empty
drwxr-xr-x   2 STC1     SYS1        8192 Jan 29 15:09 /var/run
```

Check the permissions, extended attributes, and owner of the following files:

```
$ ls -El /usr/sbin/sshd
-rwxr--r--❶ ap--  2 STC1  ❷  SYS1     8331264 Feb 25 14:30 /usr/sbin/sshd

$ ls -El /bin/ssh* /bin/scp /bin/sftp
-rwxr-xr-x  -p--❸ 2 STC1     SYS1     6041600 Feb 25 14:30 /bin/scp
-rwxr-xr-x  -p--  2 STC1     SYS1     6180864 Feb 25 14:30 /bin/sftp
-rwxr-xr-x  -p--  2 STC1     SYS1     7536640 Feb 25 14:30 /bin/ssh
-rwxr-xr-x  --s-  2 STC1     SYS1     5693440 Feb 25 14:30 /bin/ssh-add
-rwxr-xr-x  --s-  2 STC1     SYS1     5476352 Feb 25 14:30 /bin/ssh-agent
-rwxr-xr-x  --s-  2 STC1     SYS1     5918720 Feb 25 14:30 /bin/ssh-keygen
-rwxr-xr-x  --s-  2 STC1     SYS1     6070272 Feb 25 14:30 /bin/ssh-keyscan

$ ls -El /usr/lib/ssh
drwxr-xr-x        2 STC1     SYS1        8192 Oct 22  2011 IBM
-rwxr-xr-x  -p--  2 STC1     SYS1     1122304 Feb 25 14:30 sftp-server
-rwxr-xr-x  --s-  2 STC1     SYS1     3866624 Feb 25 14:30 ssh-askpass
-rwsr-xr-x  ----  2 STC1     SYS1     6418432 Feb 25 14:30 ssh-keysign
-rwxr-xr-x  aps-  2 STC1     SYS1       57344 Feb 25 14:30 zsshgss.so
```

❶ The permissions bits should match this column.
❷ The owner must be UID=0; one of your UID=0 userids should be displayed.
❸ The extended attributes should match this column. `a`="APF authorized" `p`="Program Controlled" `s`="allow shared address space"

Reference: *OpenSSH User's Guide:* "Steps for verifying the prerequisites for using OpenSSH"

1.5 Language Environment Tuning

OpenSSH uses the LE XPLINK libraries, and IBM recommends the following:

- Add SCEELPA to LPALST
- Add SCEERUN and SCEERUN2 to LNKLST
- Add SCEERUN and SCEERUN2 to LLA
- SCEERUN and SCEERUN2 must be program controlled
- Implement samples SCEESAMP(CEEWLPA) and SCEESAMP(EDCWLPA). We recommend implementing both of these as shipped.

Note: OpenSSH will still run if recommended XPLINK modules are not placed in LPA. This is something that you can defer for your next system maintenance window.

References:
- *z/OS UNIX System Services Planning* "Tuning performance"
- *Language Environment Customization* "Placing Language Environment modules in link pack and LIBPACK"

1.6 Using ICSF and /dev/random

Generation of secure random numbers is key to using OpenSSH (or any cryptographic tool). OpenSSH requires a working `/dev/random` device in order to run (the obsolete alternative `ssh-rand-helper` has been removed from OpenSSH). On z/OS Unix, `/dev/random` is provided by ICSF's `CSFRNG` service. In the past this required that you have a co-processor card, but with the "A0" or later level of ICSF (HCR77A0/A1) you don't need a co-processor card - ICSF will generate a cache of secure random numbers using CPACF instructions as appropriate.

Without `/dev/random` support, OpenSSH will fail to start with the following message:

```
FOTS1949 PRNG is not seeded. Please activate the
Integrated Cryptographic Service Facility (ICSF)
```

Assuming that ICSF is running and supports the `CSFRNG` service, all you need to do is to authorize your users to this service. For most environments, it will be acceptable to permit all users to the CSFRNG service:

```
RDEFINE CSFSERV CSFRNG UACC(NONE)
PERMIT CSFRNG CLASS(CSFSERV) ID(*) ACCESS(READ)
SETROPTS RACLIST(CSFSERV) REFRESH
```

To verify that `/dev/random` is working, issue this command from a z/OS UNIX shell and userid with normal priviledges (and CSFRNG access). This should display some random data in hex:

```
$ head /dev/random | od -x
```

Reference: *OpenSSH User's Guide:* "Using hardware support to generate random numbers"

1.7 Creating configuration files

Copy the sample configuration files to the `/etc/ssh` directory.

Note:

If you are running a previous version of OpenSSH you will want to review the differences between your current configuration files and these samples to see if any site-specific configuration options should be migrated to the new release. In particular, if you have updated `zos_ssh_config` or `zos_sshd_config` to add `CiphersSource` and `MACsSource` keywords, you will probably want to remove these. See section *Chapter 2, Exploiting crypto hardware acceleration* for more information.

Note: You must use a UID=0 userid for this:

```
$ cd /samples
$ cp -p moduli          /etc/ssh
$ cp -p ssh_config      /etc/ssh
$ cp -p sshd_config     /etc/ssh
$ cp -p zos_ssh_config  /etc/ssh
$ cp -p zos_sshd_config /etc/ssh
```

Note: All of the above files in `/etc/ssh` should be owned by a UID=0 userid and have permissions `644`:

```
-rw-r--r--   1 STC1     SYS1      242153 Jan 15 17:08 moduli
-rw-r--r--   1 STC1     SYS1          3483 Jan 15 17:08 ssh_config
-rw-r--r--   1 STC1     SYS1          4685 Jan 15 17:08 sshd_config
-rw-r--r--   1 STC1     SYS1          1158 Jan 15 17:08 zos_ssh_config
-rw-r--r--   1 STC1     SYS1          1209 Jan 15 17:08 zos_sshd_config
```

Reference: *OpenSSH User's Guide:* "Steps for creating or editing configuration files"

1.8 Creating SSHD server keys

You must generate one or more public/private key pairs that are used for authentication of your SSHD server. Each client that connects to the server will either already have one of the public keys (aka "host fingerprint") or will be required to accept your server's public key as proof of the server's identity.

For more information on SSH key authentication, see the recording of our webinar: *IBM Ported Tools for z/OS: Key Authentication*

Server keys can be stored either in protected UNIX files or in SAF/RACF keyrings. Most installations will choose to use files, which is covered below. For information on how to use SAF/RACF keyrings, see our webinar: *IBM Ported Tools for z/OS: Using Key Rings*

The following commands can be executed by a UID=0 userid to create all variants of the server key pairs. It will *not* overwrite existing keys of a given type, so it can be used safely:

```
$ cd /etc/ssh
$ ssh-keygen -A
ssh-keygen: generating new host keys: RSA1 RSA DSA ECDSA
```

```
```

This should result in four pairs of private/public key files with the following permissions, all owned by a UID=0 userid:

```
$ ls -al *key*
-rw-------   1 STC1     SYS1         668 Feb 25 15:00 ssh_host_dsa_key
-rw-r--r--   1 STC1     SYS1         606 Feb 25 15:00 ssh_host_dsa_key.pub
-rw-------   1 STC1     SYS1         227 Feb 25 15:00 ssh_host_ecdsa_key
-rw-r--r--   1 STC1     SYS1         178 Feb 25 15:00 ssh_host_ecdsa_key.pub
-rw-------   1 STC1     SYS1         981 Feb 25 15:00 ssh_host_key
-rw-r--r--   1 STC1     SYS1         646 Feb 25 15:00 ssh_host_key.pub
-rw-------   1 STC1     SYS1        1679 Feb 25 15:00 ssh_host_rsa_key
-rw-r--r--   1 STC1     SYS1         398 Feb 25 15:00 ssh_host_rsa_key.pub
```

Reference: *OpenSSH User's Guide:* "Steps for setting up server authentication when keys are stored in UNIX files"

1.9 Set up SSHD server userids

Your SSHD server will use two SAF/RACF userids (besides the actual userids that clients sign on with):

1. The privileged UID=0 userid used to start the started task (here we use "SSHDAEM")

2. The unprivileged "privilege separation" userid (this must be "SSHD", or have an alias of "SSHD")

The privileged started task userid can be an existing UID=0 userid, like OMVSKERN, but we recommend creating a new userid, defined like OMVSKERN:

```
ADDUSER SSHDAEM DFLTGRP(OMVSGRP)
  OMVS(UID(0) HOME('/') PROGRAM('/bin/sh'))
  NOPASSWORD
```

The privileged started task userid must have read access to BPX.DAEMON -

```
RDEFINE FACILITY BPX.DAEMON UACC(NONE)
PERMIT BPX.DAEMON CLASS(FACILITY) ID(SSHDAEM) ACCESS(READ)
SETROPTS RACLIST(FACILITY) REFRESH
```

Note: If your system has the FACILITY/BPX.POE defined, then the SSHDAEM userid will require READ access. If the SERVAUTH class is active, then the SSHDAEM userid will require authorization to access incoming socket connections.

Create the unprivileged "privilege separation" userid, where `ggg` is an unused groupid and `uuu` is an unused non-zero uid (you may alternatively use the `AUTOGID` and `AUTOUID` keywords if you have enabled the BPX.NEXT.USER profile) -

```
ADDGROUP SSHDG OMVS(GID(ggg))
ADDUSER SSHD DFLTGRP(SSHDG) OMVS(UID(uuu) HOME('/var/empty')
  PROGRAM('/bin/false')) NOPASSWORD
```

Reference: *OpenSSH User's Guide:* "Step for creating the sshd privilege separation user" and "Starting sshd as a stand-alone daemon"

1.10 Create SSHD server started task

The best way to start your SSHD server is by using a started task proc:

```
//SSHD PROC
//SSHD EXEC PGM=BPXBATCH,REGION=0M,TIME=NOLIMIT,
//     PARM='PGM /bin/sh -c /etc/ssh/sshd.sh'
//STDERR DD SYSOUT=*
//
```

This proc executes a shell script `/etc/ssh/sshd.sh` that you must create:

```
#!/bin/sh
export NLSPATH=$NLSPATH:/usr/lib/nls/msg/%L/%N.cat
export _EDC_ADD_ERRNO2=1
nohup /usr/sbin/sshd -f /etc/ssh/sshd_config &
sleep 1
```

This script should have permissions "700" and be owned by a UID=0 userid.

```
-rwx------   1 STC1     SYS1         141 Feb 26  2013 sshd.sh
```

The SSHD started task must be configured to start with the privileged userid "SSHDAEM" that you setup in the prior section:

```
SETROPTS GENERIC(STARTED)
RDEFINE STARTED SSHD.*
  STDATA(USER(SSHDAEM) GROUP(OMVSGRP) TRUSTED(NO))
SETROPTS RACLIST(STARTED) REFRESH
```

To start sshd, issue the following MVS command:

```
S SSHD
```

Verify that the proper userid and group were assigned to the SSHD started task by examining the system log:

```
S SSHD
$HASP100 SSHD     ON STCINRDR
IEF695I START SSHD     WITH JOBNAME SSHD     IS ASSIGNED TO USER
SSHDAEM, GROUP OMVSGRP
$HASP373 SSHD     STARTED
$HASP395 SSHD     ENDED
```

Note: like FTPD, this started task will quickly terminate, but it will spin off an OMVS address space with jobname "SSHDn".

References:

- *OpenSSH User's Guide:* "Starting sshd as a stand-alone daemon"
- *OpenSSH User's Guide:* "Ways to start sshd as a stand-alone daemon" / "Using BPXBATCH"

1.11 TCP configuration

Using the default `sshd_config` settings, SSHD listens on port 22 on all stacks. Since this is a privileged port number, only programs running as superuser are allowed to listen on this port. We recommend that for your next IPL that you also reserve this port in `PROFILE.TCPIP`, since it also serves to document usage. In the following example template, we also cause the SSHD started task to be started after TCPIP has started:

```
AUTOLOG
...
    SSHD                ; SSHD Server (STC SSHDn)
...
ENDAUTOLOG
...
PORT
...
    22 TCP SSHD* NOAUTOLOG  ; Ported Tools SSHD server
...
```

Reference: *OpenSSH User's Guide:* "Steps for creating or editing configuration files" Step 4

1.12 Verify z/OS DNS / Resolver operation

The ssh client will perform DNS lookups on target host names. Also, by default, the sshd server will lookup remote host names and check that the resolved name maps back to the IP address (warnings will be logged otherwise) . If the z/OS DNS client (the "Resolver") is not working properly, these requests might hang before timing out.

A full z/OS DNS server is not required to run OpenSSH, but most shops will want to run a `"caching-only server"` connected to their corporate DNS servers. At minimum you should at least configure the z/OS resolver so that DNS requests do not hang.

To verify that your z/OS resolver is working properly, issue the following command from a z/OS UNIX shell. Try known and unknown host names to verify that neither hang:

```
$ host www.ibm.com
EZZ8321I e3062.x.akamaiedge.net has addresses 23.67.232.41
EZZ8322I aliases: www.ibm.com, www.ibm.com.cs186.net, www.ibm.com.edgekey.net

$ host h42444.not-a-domain.com
EZZ8342I h12345.not-a-domain.com: Unknown host
```

References:

- *OpenSSH User's Guide:* "Troubleshooting" / "DNS is not configured properly"
- *z/OS Comm Svr IP Confg Guide* "The resolver" and "Domain Name System"

1.13 Configuring the syslogd daemon

SSHD server will log messages to the z/OS Communications Server **syslogd** facility using a local UNIX datagram socket (`/dev/log`).

IBM FTPD, TELNETD, IDS, and other z/OS Comm Server applications use **syslogd** for logging and tracing as well. Although some of these z/OS Comm Server applications will log to the z/OS console if **syslogd** is not running, the OpenSSH SSHD server will not. Therefore, it is important to have **syslogd** running in local mode (**-i**) so that SSHD server log messages are available.

Starting with z/OS 1.11, significant enhancements were made to **syslogd** -

- performance improvements
- improved operator interfaces
- automatic archival to MVS data sets
- ISPF syslogd viewer

The OpenSSH SSHD server by default will log all messages with `INFO` severity or higher to the local syslogd `"AUTH"` facility.

You should verify that **syslogd** is configured and started on your z/OS system and that your `/etc/syslog.conf` file is configured so that messages to the `AUTH` facility will be logged to some file. For example:

```
# log any messages to AUTH facility (sshd)
auth.* /tmp/syslogd.auth.log -X
```

References:

- *z/OS Comm Svr IP Confg Guide* "Configuring the syslog daemon"
- Presentation: *z/OS V1R11 Comm Server - syslogd enhancements*

1.14 Verify basic functionality

Before we move on, lets verify the basic functionality of your SSHD server. To do this, you will need to install an SSH client, like *PuTTY* on your workstation on a network that can connect to your z/OS system on port 22.

Note: You will want to test an ssh session to an unprivileged z/OS userid that has an OMVS segment and home directory. If you have a non UID=0, unprivileged TSO userid that can get into the TSO OMVS shell, then use that.

```
C:> putty zosuser@my.zos.host
 - or -
C:> putty zosuser@10.2.3.4
Using username "zosuser".
zosuser@my.zos.host's password: ******
/u/home/zosuser>
```

If successful, the above will place you in an SSH session with an interactive z/OS UNIX login shell. Once you have a non-TSO UNIX shell, you can use the ssh client command to connect to other hosts. For example, you can also connect to the same z/OS system using the loopback address:

```
/u/home/zosuser> ssh zosuser@127.0.0.1
The authenticity of host '127.0.0.1 (127.0.0.1)' can't be established.
RSA key fingerprint is 22:cb:9c:30:9d:98:c8:4f:45:a8:ac:00:e5:8e:62:af.
Are you sure you want to continue connecting (yes/no)? yes
zosuser@my.zos.host's password: ******
/u/home/zosuser>
/u/home/zosuser> exit
/u/home/zosuser> (back to the first connection)
```

Now is also a good time to verify your **syslogd** configuration. Look at the end of the logfile that you configured in `/etc/syslog.conf`:

```
$ tail /tmp/syslogd.auth.log
...
Feb 14 21:11:23 S0W1 sshd[67174502]: Port of Entry information retained for ...
Feb 14 21:11:24 S0W1 sshd[67174502]: Accepted password for zosuser from ...
```

2. Exploiting crypto hardware acceleration

2.1 Enabling CPACF support

Note: Apply the PTF for APAR OA54299 ("NEW FUNCTION FOR OPENSSH : CPACF SUPPORT").

With this new function, OpenSSH will use z architecture CPACF instructions directly if available for selected Cipher and MAC algorithms. This results in reduced CPU overhead as compared to using the same CPACF enabled algorithms through ICSF. In addition, configuration changes and ICSF RACF/SAF rules are not necessary. We recommend that most users will want to use this approach, which is now the default.

Note: If you have previously configured z/OS OpenSSH to use ICSF for Cipher and MAC algorithms, you may change this back to the default by removing the `CiphersSource` and `MACsSource` options from your `/etc/ssh/zos_ssh_config` and `/etc/ssh/zos_sshd_config` files. The new default for these options is `CPACF`, which will use CPACF versions of algorithms if available and OpenSSL (software) otherwise. If your setting for these options is `any`, then CPACF will be selected first (if available), and then ICSF for ICSF-compatible algorithms and then OpenSSL for the rest.

*Note:*If you have configured OpenSSH for FIPS mode, then ICSF must be used for all algorithms (this document does not support FIPS mode configurations).

2.2 Configure OpenSSH Ciphers and MACs

In this section, you will review the Cipher and MAC algorithms that your ssh client and sshd server will use. Choosing algorithms implemented via CPACF will in general greatly reduce CPU consumption when compared to other algorithms, so this is important to understand and implement correctly.

The default OpenSSH 6.4p1 Cipher and MAC algorithm names can be seen (commented out) in the sample `/etc/ssh/ssh_config` (ssh client) and `/etc/ssh/sshd_config` (sshd server) configuration files.

Following the lines are the z/OS recommended defaults, which are uncommented (active). These defaults have been selected to best optimize CPACF acceleration while maintaining a high level of compatability with non-z/OS OpenSSH implementations. For reference, these are (each entry is a single line, shown wrapped below):

```
# /etc/ssh/ssh_config changes

Ciphers aes128-ctr,aes192-ctr,aes256-ctr,aes128-cbc,aes192-cbc,aes256-cbc,
rijndael-cbc@lysator.liu.se,3des-cbc,aes256-gcm@openssh.com,
aes128-gcm@openssh.com,arcfour128,arcfour256,blowfish-cbc,cast128-cbc,arcfour

MACs hmac-sha1-etm@openssh.com,hmac-sha2-256-etm@openssh.com,
hmac-sha2-512-etm@openssh.com,hmac-sha1-96-etm@openssh.com,
hmac-sha1,hmac-sha2-256,hmac-sha2-512,hmac-sha1-96,
hmac-md5-etm@openssh.com,hmac-md5-96-etm@openssh.com,hmac-md5,hmac-md5-96,
umac-64-etm@openssh.com,umac-128-etm@openssh.com,hmac-ripemd160-etm@openssh.com,
umac-64@openssh.com,umac-128@openssh.com,hmac-ripemd160,hmac-ripemd160@openssh.com
```

These lists specify the Cipher and MAC algorithms that are supported by the ssh client and server. When an ssh client connects to an ssd server, they exchange these lists and *negotiate* which Cipher and MAC algorithm will be used for this session.

Cipher and MAC negotiation rule

The first algorithm in the client list that appears anywhere in the server list will be selected.

Configuring SSH client Ciphers and MACs

Note: The configuration in the new sample files (`/etc/ssh/ssh_config` and `/etc/ssh/sshd_config`) are a good start, since the client will try CPACF algorithms first during negotiation. Some sites will want to modify the default configurations to insure that only CPACF enabled algorithms that meet their security (strength) requirements are used.

Considering the ssh client configuration first, and mindful of the rule above, you can customize the client configuration file (`/etc/ssh/ssh_config`)), with the following considerations:

- Sites may want to list only Cipher algorithms that are implemented by CPACF with bit lengths supported on their processor (see the table in the previous section).
- Sites may choose not to support older 3des-cbc algorithms, or other CPACF or non-CPACF Ciphers that do not meet their security requirements.
- The **Host** and **Match** configuration keywords can be used in SSH client configuration files to conditionally assign different algorithm lists to specific hosts, userids, etc.

To implement a CPACF-only strategy for CPU reduction, update the `/etc/ssh/ssh_config` file, comment out the existing `MACs` and `Cipher` lines, and replace with the following (each entry is a single line, shown wrapped below):

```
# /etc/ssh/ssh_config changes

# Only allow AES ICSF/CPACF Ciphers, exclude 3des-cbc
Ciphers aes128-ctr,aes192-ctr,aes256-ctr,aes128-cbc,aes192-cbc,aes256-cbc

# Only support ICSF/CPACF SHA-1 and SHA-2 MACs:
MACs hmac-sha1-etm@openssh.com,hmac-sha2-256-etm@openssh.com,
hmac-sha2-512-etm@openssh.com,hmac-sha1-96-etm@openssh.com,
hmac-sha1,hmac-sha2-256,hmac-sha2-512,hmac-sha1-96
```

Note: Even though virtually all partner systems will support this configuration, sites with existing connections may wish to test or check SMF 119 type 94 records to see if other algorithms are in use.

Configuring SSHD server Ciphers and MACs

The negotiation rule implies that your have fewer choices for selecting Ciphers and MACs in your SSHD server configuration. To allow only CPACF accelerated Ciphers and MACs and fail otherwise, make the following changes to your `/etc/ssh/sshd_config` file:

```
# /etc/ssh/sshd_config changes

# Only support ICSF/CPACF SHA-1 and SHA-2 MACs:
MACs hmac-sha1-etm@openssh.com,hmac-sha2-256-etm@openssh.com,
hmac-sha2-512-etm@openssh.com,hmac-sha1-96-etm@openssh.com,
hmac-sha1,hmac-sha2-256,hmac-sha2-512,hmac-sha1-96

# Only allow AES ICSF/CPACF Ciphers, exclude 3des-cbc
Ciphers aes128-ctr,aes192-ctr,aes256-ctr,aes128-cbc,aes192-cbc,aes256-cbc
```

Note: Even though virtually all partner systems will support this configuration, sites with existing connections may wish to test or check SMF 119 type 95 records to see if other algorithms are in use.

Appendix A. Managing the /tmp filesystem

The `/tmp` filesystem is important to the operation of many applications that use z/OS UNIX services. Some common uses include:

- z/OS UNIX shell scripts often create temporary, transient work files. One example is setting a variable with the output of a command. Although tiny and short lived, a temporary file is required. If temp space is unavailable, then the script can fail in unexpected ways. If you will be running batch jobs that use Co:Z SFTP, IBM ssh or IBM sftp clients, you will likely be using the shell and need temp files.
- If you will be using Co:Z SFTP server, each session causes SSHD to invoke a shell script which configures the session.
- Co:Z SFTP server by default creates a log file for each session in `/tmp`. These are important to keep for some period of time:
  - The current session log file can be accessed by the remote sftp client (e.g: **get /+error.log**) to get details of a problem.
  - If there is a failure, support personnel can review the session log file for diagnostic information.
  - Trace messages, if enabled, will go to the session log.

  In many cases, installations will choose to put Co:Z SFTP server session logs in a separate zFS or HFS filesystem.

A.1 Best practices

Installations should review the references below, but here are some general suggestions:

1. Schedule a nightly job that runs the z/OS UNIX **skulker** command to clean up old files. For example, the following job (run as a superuser) uses the IBM skulker script to remove files from tmp that have not been accessed for 5 days:

```
//SKULKER EXEC PGM=COZBATCH
//STDIN   DD   *
 cd /tmp || exit 8
 /samples/skulker -R -l /dev/fd2 . 5
//
```

2. Monitor your temp filesystem(s) for full threshold conditions using the **FSFULL** mount parameter. Use the threshold messages to alert your operations personnel.

   Note: FSFULL monitoring of TFS filesystems is not supported prior to z/OS 2.1

3. Document procedures, commands, and tools to be used by personnel in the event of a full condition. Some useful commands include:

```
# display filesystem status
$ df -kP /tmp
```

```
# display 10 largest files in /tmp
$ du -aktx /tmp | sort -nr | head -10

# display pids(users) that are using a file
#   - should do this before removing a file to verify no users
$ fuser -u file

# display process/job info for a pid
$ ps -o pid,ppid,user,jobname,xasid,stime,time,comm -g <pid>

# truncate an in-use log file
#   - removing would leave an unnamed file until all using processes complete
$ cat /dev/null > file

# kill processes using a file
$ fuser -ku file

# (z/OS 2.1) list hidden in use files in a filesystem
$ zlsof -d /tmp

# (pre-z/OS 2.1) list users of deleted files
#   from IBM Tools & Toys:
#   http://www-03.ibm.com/systems/z/os/zos/features/unix/bpxa1ty2.html
$ delinuse /tmp
```

References:

- *z/OS UNIX System Services Planning* "Managing the temporary file system (TFS)"
- *z/OS UNIX System Services Command Reference* "skulker - Remove old files from a directory"