First experiences with hardware cryptographic support for OpenSSH with Linux for System z

Manfred Gnirss, Winfried Münch, Klaus Werner, Arthur Winterling

April 7, 2010

Abstract

This article summarizes our first experiences with configuration of OpenSSH using hardware cryptographic support of IBM System z. We report our first impression of performance and throughput improvements. Our positive experience indicates that where possible, you should exploit this new capability when using OpenSSH.

Contents

1 Introduction

Setting up secure and protected severs and network environments is becoming increasingly important. Besides a general need to improve security of an IT environment, which drives activities to harden servers and networks, most companies have different policies and security guidelines to be followed by the IT departments. More and more external regulations from governments and other organizations require companies to implement specific levels of security compliance. Depending on a company's business, these apply to many IT environments. One such area is protection of data traffic between systems and networks.

Common access methods for Linux® servers such as telnet, or protocols for file transfer such as FTP, should be avoided not only in Internet environments but also in internal company networks. This is because sensitive information such as passwords is transfered over the network in the clear. Using SSH and SCP increases the security because the information sent across the network is encrypted. Not only passwords, but also data is protected by encryption technologies.

By its nature, encryption of data is expensive and can have a severe impact on the performance, throughput, or CPU load of a system. IBM® System z® provides hardware encryption support that can be used to reduce the impact of expensive encryption operations. Starting with OpenSSH version 4.4, OpenSSL dynamic engine loading is supported. This enables OpenSSH to benefit from IBM System z cryptographic hardware support, if a specific option ($--with-ssl-engine$) is used during the build of the OpenSSH package. We describe our first experiences with such an OpenSSH package, and we demonstrate the value of hardware encryption support for SSH sessions[1] and file transfer with SCP and SFTP.

For our tests, we used IBM System z9® BC, IBM System z9 EC, as well as an IBM System z10™ EC and SUSE® Linux Enterprise Server for IBM System z (SLES®) 10 SP2, SLES 10 SP3 and SLES 11 GA with a pre-version of a patch provided by Novell. At the time when writing this paper, Novell has not made this patch available for general use and has not yet decided for which version of the distribution it will be provided.

2 Hardware cryptographic support of IBM System z

IBM System z provides two different types of hardware support for cryptographic operations: Central Processor Assist for Cryptographic Function (CPACF) and Crypto Express® (CEX) features.

The first type, CPACF, is incorporated in the central processors that are shipped with IBM System z. The CPACF feature was introduced with IBM System z990 and IBM System z890. The CPACF feature delivers support for symmetric encryption algorithms Data Encryption Standard (DES) and Triple DES (TDES) and hashing algorithm SHA-1. The CPACF feature incorporated in IBM System z9 also provides support for Advanced Encryption Standard (AES) with a key length of 128, as well as for SHA-256 and Pseudo Random Number Generator (PRNG). Starting with IBM System z10, also AES-192 and AES-256 are supported in the CPACF. The algorithms in the CPACF are executed synchronously with enhanced performance. These algorithms are for clear key operations (this means, the cryptographic key is provided by application software in clear format).

The second type uses additional installable Crypto Express features. For IBM System z up to IBM System z10 GA2, this is the Crypto Express2 feature. Starting with the IBM System z10 GA3, this is the Crypto Express3 feature. The Crypto Express feature can be configured either as Accelerator (CEX2A or CEX3A) or as Coprocessor (CEX2C or CEX3C). If the feature is configured as accelerator, it can perform clear key RSA operations at very high speed. If configured as Coprocessor, it can perform symmetric as well as asymmetric operations (RSA) in clear key and also in secure key mode. Note, the operations executed by the Crypto Express feature are performed asynchronously outside the central processor. This means work is partially off-loaded and CPU cycles for cryptographic operations are reduced, compared with the equivalent execution of a pure software implementation.

[1] Note, the commercial vendor product SSH Tectia® Server for Linux on IBM System z also automatically uses hardware acceleration of cryptographic operations with the IBM-provided cryptographic hardware CPACF.

To benefit from the CPACF, you must install LIC internal feature 3863 (Crypto Enablement feature), which is available free of charge (see also [1]). By default, the IBM System z is delivered to customers without this feature, unless it is ordered explicitly by the customer. The installation of this feature at a future time is non-disruptive.

We recommend that you install the Crypto Enablement feature, even if you do not intend to use the Crypto Express2 or Crypto Express3 feature, because there is still a benefit from an active CPACF.

2.1 Verification of installed LIC 3863 using the HMC or SE

You can check if the CPACF is enabled in your environment by the dialogues provided on the Support Element (SE) or Hardware Management Console (HMC). In the *CPC Details* panel you can find "CP Assist for Crypto functions: Installed" (see Figure 1) or "CP Assist for Crypto functions: Not installed".

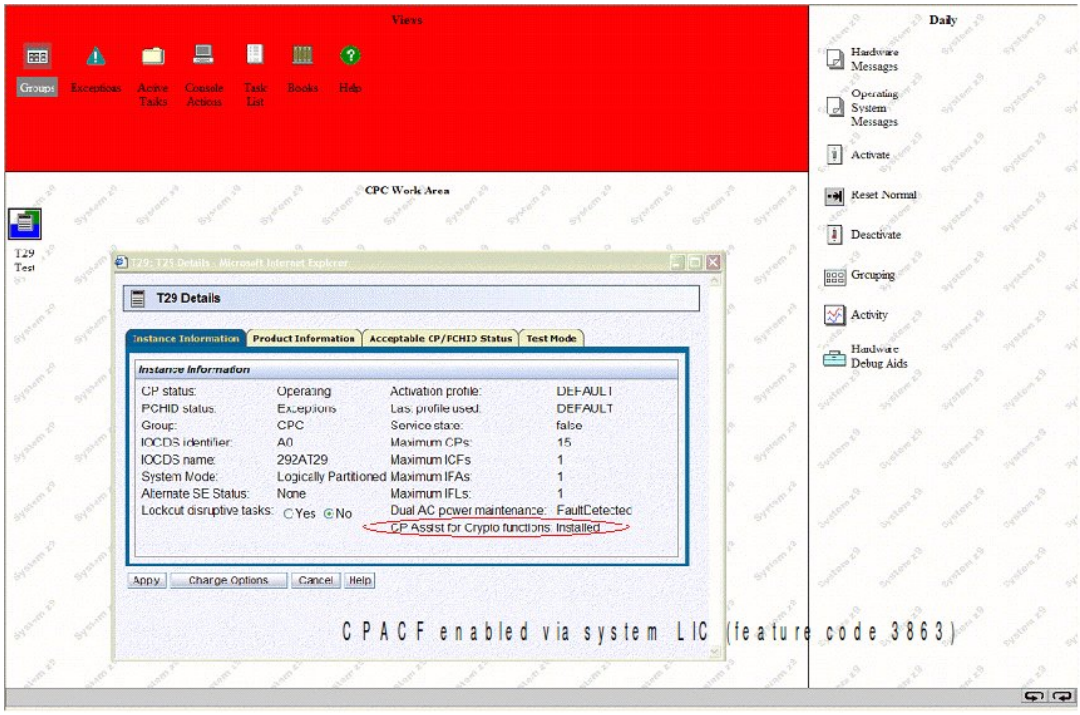

Figure 1: Feature Code 3863 (CPACF) installed

2.2 Verification of installed LIC 3863 using a Linux command

A Linux for System z user can check easily whether the Crypto Enablement feature is installed and which algorithms are supported in hardware. The command *icainfo* displays which CPACF functions are supported by the libica library on your server. Starting with libica Version 1.3.9 this command is available after the installation of the library.

If the Crypto Enablement feature 3863 is not installed, you will see that only SHA is supported and all other algorithms are not available in CPACF (see Example 1).

```
tsin3300:/usr/lib64> icainfo
The following CP Assist for Cryptographic Function (CPACF) operations are
supported by libica on this system:
SHA-1:      yes
SHA-256:    yes
SHA-512:    yes
```

```
DES:        no
TDES-128:   no
TDES-192:   no
AES-128:    no
AES-192:    no
AES-256:    no
PRNG:       no
```

Example 1: Response of icainfo command on IBM System z10, if LIC 3863 is not installed

If the Crypto Enablement feature 3863 is installed, you will see that besides SHA, other algorithms are available with hardware support. In Example 2 issued on a z9®, you can see that AES-192 and AES-256 are not supported in CPACF on the running system, while in the example shown in Example 3 issued on a z10™, you can see that AES-192 and AES-256 support is available.

```
tmcc-123-180:/usr/lib64> icainfo
The following CP Assist for Cryptographic Function (CPACF) operations are
supported by libica on this system:
SHA-1:      yes
SHA-256:    yes
SHA-512:    no
DES:        yes
TDES-128:   yes
TDES-192:   yes
AES-128:    yes
AES-192:    no
AES-256:    no
PRNG:       yes
```

Example 2: Encryption algorithms supported in CPACF of IBM System z9

```
gnirss@tmcc-123-252:~> icainfo
The following CP Assist for Cryptographic Function (CPACF) operations are
supported by libica on this system:
SHA-1:      yes
SHA-256:    yes
SHA-512:    yes
DES:        yes
TDES-128:   yes
TDES-192:   yes
AES-128:    yes
AES-192:    yes
AES-256:    yes
PRNG:       yes
```

Example 3: Encryption algorithms supported in CPACF of IBM System z10

3 Configuration of Crypto Express feature for Linux for IBM System z

If you have a Crypto Express2 (CEX2) or a Crypto Express3 (CEX3) feature in your System z, you can also benefit from hardware support for the RSA handshake while opening a SSH session.

For information about how to configure the LPAR Activation Profile, and how to enable access to the CEX2 feature for a Linux system running in a z/VM environment on IBM System z9, see Chapter 6 of [2] for details. For comparable information for IBM System z10 with CEX2 or CEX3, see Chapter 8 of [3]. In [4] and [5] information about how to work with the SE and the HMC can be found. Detailed

background information about IBM System z cryptographic hardware capabilities can be found in [6] and [7].

4 Linux cryptographic architecture used by OpenSSH

Here is a small overview of how OpenSSH accesses the hardware cryptographic support provided by IBM System z (see Figure 2). OpenSSH uses OpenSSL to perform the cryptographic requests. If the OpenSSH package is built using the option $--with-ssl-engine$, then the OpenSSL library can use available engines and load them dynamically. In a System z environment, you can install the ibmca engine and configure OpenSSL for dynamic engine loading. In this case, OpenSSL does not perform the encryption requests by itself, but passes them to the ibmca engine. The ibmca engine uses the library libica to handle the requests. The libica library is aware of which algorithms are supported by the underlying hardware (if installed and available) CPACF or Crypto Express feature. If an algorithm is supported by the underlying hardware, the libica library passes the request to the cryptographic hardware. If an algorithm is not supported by the underlying hardware, the libica library executes the algorithm in software as a fall-back[2]. The underlying virtualization layer of z/VM has no impact on the cryptographic architecture inside the Linux server. The only consideration here is that z/VM can virtualize the access to the Crypto Express feature. You need to adapt the z/VM directory, if you intend to access the Crypto Express feature from Linux (see Chapter 6 of [2]).

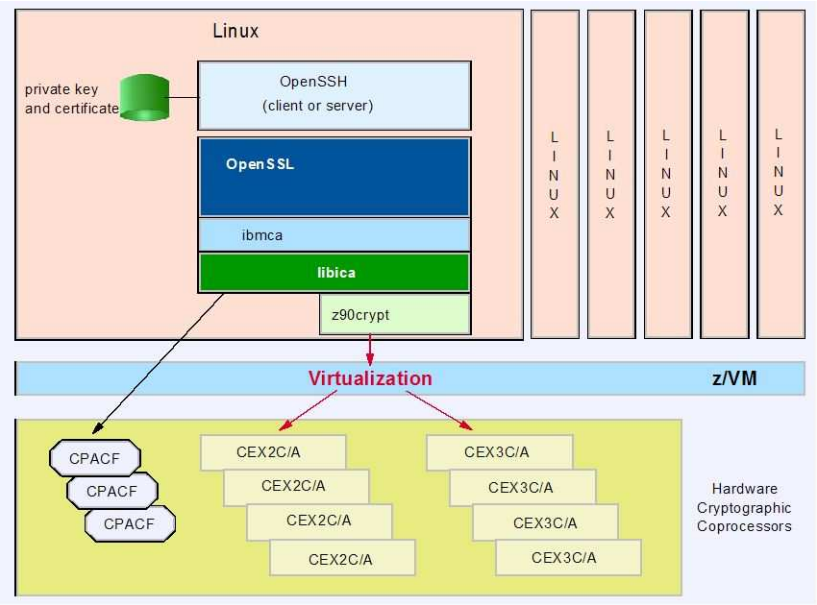

Figure 2: Linux for System z environment for hardware cryptographic support for OpenSSH

The following software and driver packages need to be installed on Linux for System z to enable OpenSSH to benefit from hardware cryptographic support on IBM System z.

- openssh
- openssl
- openssl-ibmca

[2]Starting with libica V2, libica uses the OpenSSL library for execution of cryptographic requests, if software fall-back is necessary.

- libica
- z90crypt

All these packages are part of the Linux for System z distribution. Some of them might be already installed with your initial set up. Probably, the packages openssl-ibmca and libica have to be installed separately.

5 Preparation of OpenSSL for ibmca engine use

We start with a SUSE Linux SLES 11 GA environment in which OpenSSL is installed, but dynamic engine loading support is not activated. The required libibmca libraries which contains the engine ibmca and the libica library are not yet installed (see Example 4).

```
gnirss@tmcc-123-180:~> cat /proc/cpuinfo
vendor_id       : IBM/S390
# processors    : 2
bogomips per cpu: 6868.00
features        : esan3 zarch stfle msa ldisp eimm dfp
processor 0: version = FF,  identification = 01A75E,  machine = 2094
processor 1: version = FF,  identification = 01A75E,  machine = 2094

gnirss@tmcc-123-180:~> uname -a
Linux tmcc-123-180 2.6.27.19-5-default #1 SMP 2009-02-28 04:40:21 +0100
s390x s390x s390x GNU/Linux

gnirss@tmcc-123-180:/> cat /etc/SuSE-release
SUSE Linux Enterprise Server 11 (s390x)
VERSION = 11
PATCHLEVEL = 0


gnirss@tmcc-123-180:~> rpm -qa |grep openssl
libopenssl0_9_8-32bit-0.9.8h-30.11
openssl-certs-0.9.8h-25.14
openssl-0.9.8h-30.11
libopenssl0_9_8-0.9.8h-30.11

gnirss@tmcc-123-180:~> rpm -qa | grep libica
gnirss@tmcc-123-180:~>

gnirss@tmcc-123-180:~> rpm -qa | grep ibmca
gnirss@tmcc-123-180:~>
```

Example 4: OpenSSL environment without dynamic engine loading

It can also be verified by an *openssl engine* command (see Example 5) that the ibmca engine for IBM System z is not yet available in our environment.

```
gnirss@tmcc-123-180:~> openssl engine
(dynamic) Dynamic engine loading support

gnirss@tmcc-123-180:~> openssl engine -c
(dynamic) Dynamic engine loading support
```

Example 5: OpenSSL engine interface

In addition, to compare the effect of hardware encryption by CPACF on your System z you can invoke an *openssl speed* test program and store the result for later comparison with the environment including dynamic engine loading.

```
gnirss@tmcc-123-180:~> openssl speed -evp des-ede3-cbc
--- some lines not displayed ---
The 'numbers' are in 1000s of bytes per second processed.
type             16 bytes     64 bytes    256 bytes   1024 bytes   8192 bytes
des-ede3-cbc      8096.65k     8507.63k     8615.64k     8648.86k     8652.29k
```

Example 6: OpenSSL speed test program: des-ede3-cbc *in software on a IBM System z9 EC*

```
gnirss@tmcc-123-180:~> openssl speed -evp aes-128-cbc
--- some lines not displayed ---
The 'numbers' are in 1000s of bytes per second processed.
type             16 bytes     64 bytes    256 bytes   1024 bytes   8192 bytes
aes-128-cbc      20337.78k    22705.34k    23403.54k    23546.18k    23565.01k
```

Example 7: OpenSSL speed test program: aes-128-cbc *in software on a IBM System z9 EC*

The results in Example 6 and 7 of the small *openssl speed* test program show the expected behavior indicating that AES is a faster algorithm than TDES.

To enable dynamic engine loading for Linux for System z, you perform the following three steps:

1. Install and ensure that the *ibmca* engine is installed

   ```
   gnirss@tmcc-123-180:~> rpm -qa | grep ibmca
   openssl-ibmca-1.0.0-104.10
   openssl-ibmca-32bit-1.0.0-104.10
   ```

2. Install and ensure *libica* library is installed

   ```
   gnirss@tmcc-123-180:~> rpm -qa | grep libica
   libica-1_3_9-1.3.9-6.6
   libica-1_3_9-32bit-1.3.9-6.6
   ```

3. Customize the OpenSSL configuration file to use dynamic engine loading

Note that you need appropriate privileges in the system to perform these steps.

Because the libica library and the engine ibmca support are shipped in separate RPM packages, you need to install them in addition to OpenSSL.

As part of the *openssl-ibmca* package there is an *openssl.cnf.sample* file (see Example 8). You can use this file to enable the use of the ibmca engine for applications that are aware of the dynamic engine support of OpenSSL.

```
#
# OpenSSL example configuration file. This file will load the IBMCA engine
# for all operations that the IBMCA engine implements for all apps that
# have OpenSSL config support compiled into them.
#
# Adding OpenSSL config support is as simple as adding the following line to
# the app:
#
# #define OPENSSL_LOAD_CONF      1
#
openssl_conf = openssl_def

[openssl_def]
```

```
engines = engine_section

[engine_section]

foo = ibmca_section

[ibmca_section]
dynamic_path = /usr/lib64/engines/libibmca.so
engine_id = ibmca
default_algorithms = ALL
#default_algorithms = RAND,RSA
init = 1
```

Example 8: The openssl.cnf.sample file for ibmca

To enable the engine *ibmca*, the OpenSSL configuration file has to be adapted.

To customize the OpenSSL configuration to enable dynamic engine loading for ibmca, perform the following 4 steps:

1. Do not forget to make a backup of the configuration file before you change it.
2. Append the content of the sample file */usr/share/doc/packages/openssl-ibmca/openssl.cnf.sample* (see Example 8) to the existing *openssl.cnf* file on the host.
3. Move the following line

   ```
   openssl_conf = openssl_def
   ```

   of the appended part to the top of the configuration file. The configuration file now looks as shown in Example 9. Note that this might not be mentioned in the README file of the current openssl-ibmca package.
4. Check the value of the *dynamic_path* variable and if necessary change it to the correct path, which is dependent on the Linux distribution in use.

```
#
# OpenSSL example configuration file.
# This is mostly being used for generation of certificate requests.
#

# This definition stops the following lines choking if HOME isn't
# defined.
HOME                    = .
RANDFILE                = $ENV::HOME/.rnd
# --- next line: enable dynamic engine ibmca - MG 14.12.2009 ---
openssl_conf = openssl_def
# Extra OBJECT IDENTIFIER info:
#oid_file               = $ENV::HOME/.oid
oid_section             = new_oids

--- some lines not displayed ---

# This really needs to be in place for it to be a proxy certificate.
proxyCertInfo=critical,language:id-ppl-anyLanguage,pathlen:3,policy:foo
#
# OpenSSL example configuration file. This file will load the IBMCA engine
# for all operations that the IBMCA engine implements for all apps that
```



```
# have OpenSSL config support compiled into them.
#
# Adding OpenSSL config support is as simple as adding the following line to
# the app:
#
# #define OPENSSL_LOAD_CONF      1
#
# --- next line kept as comment, while moved to top of file - MG 14.12.2009 ---
#openssl_conf = openssl_def

[openssl_def]
engines = engine_section

[engine_section]

foo = ibmca_section

[ibmca_section]
dynamic_path = /usr/lib64/engines/libibmca.so
engine_id = ibmca
default_algorithms = ALL
#default_algorithms = RAND,RSA
init = 1
```

Example 9: File /etc/ssl/openssl.cnf with dynamic engine loading for ibmca

A first check of whether the dynamic engine support for ibmca is enabled, can be done using the *openssl engine* command as shown in Example 10.

```
gnirss@tmcc-123-180:/etc/ssl> openssl engine
(dynamic) Dynamic engine loading support
(ibmca) Ibmca hardware engine support

gnirss@tmcc-123-180:/etc/ssl> openssl engine -c
(dynamic) Dynamic engine loading support
(ibmca) Ibmca hardware engine support
 [RSA, DSA, DH, RAND, DES-ECB, DES-CBC, DES-EDE3, DES-EDE3-CBC, AES-128-ECB,
AES-128-CBC, AES-192-ECB, AES-192-CBC, AES-256-ECB, AES-256-CBC, SHA1, SHA256]
```

Example 10: OpenSSL engine interface with dynamic engine ibmca

For later tests, we simply enable and disable dynamic engine loading support by a small change to the *openssl.cnf* file.

To disable dynamic engine loading of ibmca, comment out the line

```
openssl_conf = openssl_def
```

at the top of the file.

To enable dynamic engine loading of the engine *ibmca*, uncomment the line again.

```
gnirss@tmcc-123-180:/etc/ssl> openssl speed -evp des-ede3-cbc
--- some lines not displayed ---
The 'numbers' are in 1000s of bytes per second processed.
type             16 bytes     64 bytes    256 bytes   1024 bytes   8192 bytes
des-ede3-cbc     36628.07k   100230.85k   182038.60k   230052.05k   248228.60k
```

Example 11: OpenSSL speed test program: des-ede3-cbc *with ibmca engine support (CPACF) on a z9 EC*

```
tmcc-123-180:/usr/lib64> openssl speed -evp aes-128-cbc
--- some lines not displayed ---
```

Table 1: Throughput for 8 KB blocks encrypted with openssl speed -evp <cipher>

Server	Cipher	Without dynamic engine [MB/s]	With dynamic engine ibmca [MB/s]
z9 BC	des-ede3-cbc	7.1	204.0
z9 BC	aes-128-cbc	20.3	272.5
z9 EC	des-ede3-cbc	8.7	248.2
z9 EC	aes-128-cbc	23.6	329.6
z10 BC	des-ede3-cbc	9.7	245.4
z10 BC	aes-128-cbc	34.5	806.8
z10 BC	aes-192-cbc	29.7	725.1
z10 BC	aes-256-cbc	26.0	656.1
z10 EC	des-ede3-cbc	14.2	310.1
z10 EC	aes-128-cbc	48.5	1012.1
z10 EC	aes-192-cbc	41.4	905.6
z10 EC	aes-256-cbc	35.9	819.6

```
The 'numbers' are in 1000s of bytes per second processed.
type             16 bytes     64 bytes    256 bytes   1024 bytes   8192 bytes
aes-128-cbc      36255.54k   109844.17k   221510.98k   297934.34k   329578.68k
```

Example 12: OpenSSL speed test program: aes-128-cbc *with ibmca engine support (CPACF) on a z9 EC*

If you compare the results shown in Example 11 and 12 with the prior results shown in Example 6 and 7 of the small *openssl speed* test program, you can see a huge performance and throughput increase for TDES and for AES. This improvement results from the use of CPACF. On an IBM System z10, there are considerably higher throughput values for AES, because not only the clock speed of the z10 is increased, but also the hardware support of CPACF for AES has been improved. In Table 1 is a collection of some results of the small test using *openssl speed -evp <cipher>*, which shows the effect of the CPACF on encryption speed with System z.

6 Installation of OpenSSH patch

To benefit from hardware support for OpenSSH, you need a minimum level of OpenSSH and this package must be built using the option *−−with−ssl−engine*. This option enables OpenSSH to request the dynamic engine loading support for ibmca engine by OpenSSL. Until such a package is part of an official Linux distribution, customers who have an account for the Novell Customer Center and are entitled to service for s390x, could ask Novell whether such an updated package of OpenSSH is already available.

After such an updated package becomes available, it would also be possible for customers to verify whether the OpenSSH package has been built using the option *−−with−ssl−engine* by checking the SPEC-file and the built options. For this purpose, examine the related src.rpm which can be downloaded from download.novell.com.

The tests for this paper are executed with a pre-version of such an OpenSSH package. The installation of this package with the build options

```
yast -i openssh-5.1p1-41.24.708.3.PTF.552126.s390x.rpm
```

needs root privileges on the system.

A reboot of the system or restarting of the SSHD daemon is not necessary. At a minimum, we verify whether the updated package is installed (see Example 13).


```
gnirss@tmcc-123-180:~> rpm -qa | grep OpenSSH
OpenSSH-5.1p1-41.24.708.3.PTF.552126
OpenSSH-askpass-5.1p1-41.24
OpenSSH-5.1p1-41.24
```

Example 13: OpenSSH Patch installed

7 Verification for usage of library libica to access CPACF

After setting up and configuring OpenSSH for the use of the libica library and CPACF, you will want to check whether CPACF is really invoked for encryption when a cipher with an algorithm supported by CPACF is used. Besides the indirect proof by observing better performance, there are several options to verify whether the libica library and CPACF are used for encryption of data when OpenSSH is used.

7.1 Hardware tracing of CPACF instructions

Starting with IBM System z10 EC, tracing of CPACF instructions is available. This function is for restricted use and only available in PEMODE, and therefore not for general customer use.

7.2 Verification of loaded library libica

The easiest way to check whether IBM System z hardware support is used during execution of the tests, is to check whether the library libica and ibmca are loaded into memory. We know that both libica and ibmca are present in memory only during execution of a program that needs services from them. Therefore, we use a second SSH session to the Linux server and the *lsof* command to check whether a library is loaded or not. Note that for this test we enable dynamic engine loading support for engine ibmca after our working SSH sessions have been opened, and just before we open another SSH session or issue an SCP command (for the working session we are not using dynamic engine loading).

As soon as dynamic engine loading support is enabled, we can observe during execution of new SSH and SCP commands that both libraries, *libibmca* and *libica* are loaded into the memory (see Example 14).

```
gnirss@tmcc-123-180:~> sudo lsof | grep ibmca
ssh   8732 gnirss mem  REG  8,1  36320  217405 /usr/lib64/engines/libibmca.so.0.0.0
sshd  8733   root mem  REG  8,1  36320  217405 /usr/lib64/engines/libibmca.so.0.0.0
sshd  8736 gnirss mem  REG  8,1  36320  217405 /usr/lib64/engines/libibmca.so.0.0.0
gnirss@tmcc-123-180:~> sudo lsof | grep libica
ssh   8732 gnirss mem  REG  8,1 102560  105872 /usr/lib64/libica-1.3.9.so
sshd  8733   root mem  REG  8,1 102560  105872 /usr/lib64/libica-1.3.9.so
sshd  8736 gnirss mem  REG  8,1 102560  105872 /usr/lib64/libica-1.3.9.so
```

Example 14: Libraries libica and libibmca are loaded into memory

After the SSH and SCP commands have finished, the libraries *libica* and *libibmca* are freed, and are no longer in memory (see Example 15).

```
gnirss@tmcc-123-180:~> sudo lsof | grep ibmca
gnirss@tmcc-123-180:~> sudo lsof | grep libica
gnirss@tmcc-123-180:~>
```

Example 15: Libraries libica and libibmca are not in memory

7.3 Outlook: Use counter of executed cryptographic requests in libica

The most definite way to prove that OpenSSH uses CPACF is to count the requests executed by CPACF while running an OpenSSH session. Because tracing of the executed instructions in CPACF requires complex setup and is restricted (see chapter 7.1), a method inside the Linux system itself is more appropriate for our purposes. Starting with libica Version 2, there is a small tool *icastats* included, which is used to query the number of executed encryption requests handled by the libica library. The tool distinguishes between requests that can be executed in CPACF hardware and requests executed as software fall-back in the libica library (because either the algorithm is not supported by the current CPACF or CPACF is not enabled (see chapter 2)). The libica V2 library is not yet included in SUSE Linux SLES 10 SP2, SP3 or SLES 11 GA. Therefore this tool cannot yet be used in any supported environment at the time when this paper was written.

For a temporary test, we downloaded the source of the libica V2 library from sourceforge.net and built our own libica V2 library. The result is as expected. While copying data[3] using SCP with TDES cipher (as shown in Example 19), the *icastats* tool shows clearly that encryption and decryption requests for TDES are passed to CPACF by the *libica* library (see Example 16). When we use the AES cipher, we can see that the counters for the AES encryption and decryption requests processed in CPACF increased (see Example 17).

We performed this test on a Linux server running on a System z9 with no access to a Crypto Express feature. Therefore the RSA requests during the session initialization are performed in the libica library as software fall-back. This is also illustrated in the Examples 16 and 17, the counters for MOD EXPO and RSA CRT requests executed in software by the libica library increased.

```
gnirss@tmcc-123-180:/usr/lib64> icastats
  function  | # hardware  | # software
------------+-------------+------------
      SHA1  |         469 |          0
    SHA224  |           0 |          0
    SHA256  |          40 |          0
    SHA384  |           0 |          0
    SHA512  |           0 |          0
    RANDOM  |           3 |          0
  MOD EXPO  |           0 |          5
   RSA CRT  |           0 |          1
   DES ENC  |           0 |          0
   DES DEC  |           0 |          0
  3DES ENC  |       14225 |          0
  3DES DEC  |       28450 |          0
   AES ENC  |           0 |          0
   AES DEC  |           0 |          0
```

Example 16: Using icastats of libica Version 2 to verify CPACF usage, while file transfer with TDES

```
gnirss@tmcc-123-180:/usr/lib64> icastats
  function  | # hardware  | # software
------------+-------------+------------
      SHA1  |         467 |          0
    SHA224  |           0 |          0
    SHA256  |          40 |          0
    SHA384  |           0 |          0
    SHA512  |           0 |          0
    RANDOM  |           3 |          0
  MOD EXPO  |           0 |          5
```

[3]For this test dynamic engine loading is enabled after the SSH working sessions to the system are already opened. An increase of the counters shown by icastats is therefore based on the copying of data with SCP and not based on the SSH working session.


```
  RSA CRT |            0 |            1
  DES ENC |            0 |            0
  DES DEC |            0 |            0
 3DES ENC |            0 |            0
 3DES DEC |            0 |            0
  AES ENC |        14248 |            0
  AES DEC |        28496 |            0
```

Example 17: Using icastats of libica Version 2 to verify CPACF use, while file transfer with AES

As soon as the libica V2 library becomes part of official Linux for System z distributions, you can use *icastats* for a fast verification of libica and CPACF use when running any application that uses cryptographic operations.

8 First experiences with OpenSSH and hardware support for encryption

Disclaimer:
All numbers presented in the following section are not the result of official benchmark tests. These results may not be reproducible in any other environment, and they are not intended to be used for any sizing estimates. Note that all our Linux servers are running as guests in a shared z/VM environment.

Many factors influence the overall performance of data transfer with SSH, such as network infrastructure, TCP/IP stack, disk performance, selected cipher suite, effort for encryption, and others. To get a feel for the effect of using dynamic engine loading on an IBM System z and System z hardware cryptographic support for SSH, we have run the following different scenarios:

- A file transfer to the localhost
- A file transfer from a host down to the PC
- A file transfer between Linux guests inside a z/VM
- Influence of using a Crypto Express feature for SSH
- Test of SFTP
- Test of rsync over SSH

8.1 File transfer with SCP to the localhost

To minimize the impact of the external network infrastructure, we perform a file transfer using SCP inside the Linux server using localhost as the target host name. To reduce disk I/O, we copy the transfered file to /dev/null. For the test we are using a source file of approximately 200 MB in size and a Linux server with SUSE SLES 11 GA, running on an IBM System z9 EC with two active CPUs (see Example 18). Because we are repeating the test multiple times and because the memory of our Linux server is large enough, the source file to be transferred resides in the file system cache.

```
gnirss@tmcc-123-180:~> cat /proc/cpuinfo
vendor_id       : IBM/S390
# processors    : 2
bogomips per cpu: 6868.00
features        : esan3 zarch stfle msa ldisp eimm dfp
processor 0: version = FF,  identification = 01A75E,  machine = 2094
processor 1: version = FF,  identification = 01A75E,  machine = 2094
```

```
gnirss@tmcc-123-180:/> cat /etc/SuSE-release
SUSE Linux Enterprise Server 11 (s390x)
VERSION = 11
PATCHLEVEL = 0


gnirss@tmcc-123-180:~/test11122009> ls -al testdata.txt
-rw-r--r-- 1 gnirss users 205468776 16. Dez 11:19 testdata.txt
```

Example 18: Environment

The following command is used for the test:

```
time scp -c <cipher> testdata.txt gnirss@localhost:/dev/null
```

We are using the *time* function to automatically obtain the information about elapsed time, user time and system time. The option *-c* specifies a cipher[4] for encryption, such as *3des-cbc*, *aes-128-cbc*, or *aes-192-cbc*. If the option *-c* is omitted, a default cipher[5] is used. Example 19 shows an example of SCP using TDES and having dynamic engine loading active.

```
gnirss@tmcc-123-180:~/test11122009> time scp -c 3des-cbc
testdata.txt gnirss@localhost:/dev/null
Password:
testdata.txt                  100%  196MB  32.7MB/s   00:06

real    0m7.462s
user    0m2.180s
sys     0m0.816s
```

Example 19: SCP to local host example

In Table 2 we have collected some results of executing SCP using different ciphers with and without having dynamic engine loading active. In our environment, when the encryption is executed using the CPACF (by default, TDES and AES-128) you can see a considerable improvement in the elapsed time (columns with heading *real*). However, the elapsed execution time is not accurate because it includes the time taken to type the password on the keyboard. More interesting are the values in the columns with heading *user*. The user time for the SCP command decreases dramatically when CPACF is used. There is also an increase in throughput (columns with heading *tp*). There is nothing particular to say about the system time (columns with heading *sys*). Because these tests are performed on a IBM System z9, cipher AES-192 and higher are not supported using CPACF. With dynamic engine loading support active, the encryption for AES-192 is performed within the libica library as software fall-back instead of OpenSSL. Therefore, no significant change is visible for cipher AES-192 when using dynamic engine loading of ibmca.

We repeat the test on a z10 EC and get results as shown in Table 3.

8.2 Download file with SCP from Linux host to PC

The second scenario is a download from the z9 Linux host to the local PC. The PC is connected to the Linux server over our general network infrastructure. For this test, the network infrastructure can be seen as complex and unknown. We want to check whether we can see any effect from using CPACF on the Linux host. The host environment is the same as used in chapter 8.1.

Example 20 shows how we download the data from the host to the PC's current directory.

```
gnirss@gnirss-tmcc:~$ time scp -c aes128-cbc
gnirss@9.152.123.180:/home/gnirss/test11122009/testdata.txt .
```

[4] At the time of writing this article, we had not completed testing the cipher aes256-cbc.

[5] If no cipher is explicitly specified, the default depends on the configuration settings for SSH and SSHD (see chapter 9).

Table 2: SCP to localhost on a IBM System z9 EC (default MAC=MD5)

Cipher	Without dynamic engine				With dynamic engine ibmca				
	real [s]	user [s]	sys [s]	tp [MB/s]	real [s]	user [s]	sys [s]	tp [MB/s]	encryption via
default	15.6	9.8	0.6	15.1	7.7	2.0	0.8	39.2	CPACF
3des-cbc	33.6	25.2	0.9	6.3	7.6	2.2	0.8	39.2	CPACF
aes128-cbc	15.2	9.8	0.6	15.1	7.4	1.9	0.7	39.2	CPACF
aes192-cbc	16.8	11.4	0.6	13.1	16.9	12.0	0.6	14.0	sw in libica

Table 3: SCP to localhost on a IBM System z10 EC (default MAC=MD5)

Cipher	Without dynamic engine				With dynamic engine ibmca			
	real [s]	user [s]	sys [s]	tp [MB/s]	real [s]	user [s]	sys [s]	tp [MB/s]
default	10.1	5.3	0.5	28.0	5.5	0.9	0.5	98.0
3des-cbc	19.9	15.2	0.5	12.3	6.3	1.4	0.5	65.3
aes128-cbc	11.0	5.2	0.5	28.0	6.1	0.9	0.4	98.0
aes192-cbc	11.6	6.2	0.5	24.5	5.4	0.9	0.4	98.0

```
Password:
testdata.txt                                  100%  196MB  10.9MB/s   00:18

real    0m21.301s
user    0m5.516s
sys     0m2.144s
```

Example 20: SCP download file from host to PC

In our case, enabling or disabling dynamic engine loading on the Linux host does not change the behavior observed on the PC. The time needed for downloading the 200 MB data is in both cases always approximately 21 seconds. This means that for this test, we are facing some kind of a network bottleneck or hard disk limitation. Using CPACF on the host neither speeds up the file transfer nor increases the throughput. But checking the system load on the host using *top* command while the download is active demonstrates an effect. In our case, we have a CPU consumption of approximately 16% for *sshd* server if CPACF is used (see Example 21) and approximately 69% if CPACF is not used (see Example 22). Please note, the values from the *top* command are changing while the test is running, so the numbers presented here can only be used as a rough indicator. However, we can see clearly that the CPU load is decreased.

```
Tasks:  67 total,   1 running,  66 sleeping,   0 stopped,   0 zombie
Cpu(s):  3.6%us,  1.0%sy,  0.0%ni, 89.4%id,  0.0%wa,  0.1%hi,  3.1%si,  2.7%st
Mem:    506580k total,   484560k used,    22020k free,    56008k buffers
Swap:   963860k total,       60k used,   963800k free,   376848k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
21143 gnirss    20   0 13448 5792 1352 S   16  1.1   0:01.78 sshd
21144 gnirss    20   0  5620 1624 1220 S    2  0.3   0:00.28 scp
    1 root      20   0  1116  384  328 S    0  0.1   0:05.08 init
--- some lines not displayed ---
```

Example 21: Output of top command while SCP from host to PC: dynamic engine loading is active

Table 4: Throughput of SCP between Linux guests inside one z/VM using OSA Express2 SX depending on dynamic engine loading support on an IBM System z9 BC (default cipher AES-128 and MAC MD5)

Dynamic engine loading support active in	Throughput [MB/s]
none	6.5
sender only	9.4
receiver only	9.4
sender and receiver	20.0

```
top - 16:32:03 up 3 days, 23:00,  3 users,  load average: 0.40, 0.12, 0.03
Tasks:  64 total,   2 running,  62 sleeping,   0 stopped,   0 zombie
Cpu(s): 24.5%us,  1.7%sy,  0.0%ni, 64.7%id,  0.1%wa,  0.1%hi,  4.4%si,  4.4%st
Mem:    506580k total,   480048k used,    26532k free,    54872k buffers
Swap:   963860k total,       60k used,   963800k free,   376812k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
20959 gnirss    20   0 12388 4900 1340 R   69  1.0   0:09.94 sshd
20960 gnirss    20   0  5620 1624 1220 S    2  0.3   0:00.35 scp
    1 root      20   0  1116  384  328 S    0  0.1   0:05.04 init
--- some lines not displayed ---
```

Example 22: Output of top command while SCP from host to PC: dynamic engine loading is not active

8.3 File transfer with SCP between Linux guests inside of one z/VM

The next tests are performed on an IBM System z9 BC. We are sending data from one Linux guest to a second Linux guest inside one LPAR with z/VM. The Linux guests are connected to the network using a shared OSA Express2® SX adapter, meaning that the data flow is with the network adapter, and does not use any external network infrastructure (cables). The z/VM system has only one CPU, which is used by z/VM itself and the Linux guests. The Linux system is SUSE SLES 11, and we transfer a file using the *scp* command.

First, we want to check the influence of dynamic engine loading support if none of, one, or both the Linux guests have hardware cryptographic support available for SSH.

We transfer a file of approximately 500 MB in size from one guest to the second guest. The file is copied from a temporary file system residing in memory and is transfered to the target Linux system also on a temporary file system. This also minimizes the influence of disk I/O. In this first test we are using the defaults (cipher is *aes128-cbc* and MAC is MD5) of our SCP configuration (see Example 23):

```
zvmgast7:/tmp # scp meter zvmnet8:/tmp
Password:
meter                                         100%  500MB   6.5MB/s   01:17
```

Example 23: SCP between two Linux guests on a z9 BC

We enable dynamic engine loading for ibmca in the Linux guests and check the effect on the throughput. In Table 4 we show the results for the throughput in our environment if none of the guests, only the sender, only the receiver, or both have dynamic engine loading support active. We can clearly see the benefit of using CPACF support on throughput in this scenario. In particular, we see a huge throughput increase when both guests benefit from dynamic engine support.

Now we check the influence of dynamic engine support using different ciphers specified with the "*-c*" option of the *scp* command when we transfer data (file size approximately 200 MB) between Linux guests

Table 5: SCP between Linux guests inside one z/VM on IBM System z9 BC using OSA Express2 SX

Cipher	Without dynamic engine				With dynamic engine ibmca				
	real [s]	user [s]	sys [s]	tp [MB/s]	real [s]	user [s]	sys [s]	tp [MB/s]	encryption via
default	33.2	12.3	1.7	6.5	11.7	2.4	1.9	22.2	CPACF
3des-cbc	69.4	31.1	2.2	3.0	11.9	2.7	1.9	20.0	CPACF
aes128-cbc	33.5	12.5	1.5	6.5	11.3	2.4	2.0	22.2	CPACF
aes192-cbc	37.2	14.4	1.7	5.7	37.3	14.6	1.8	5.6	sw in libica

Table 6: SCP between Linux guests inside one z/VM on IBM System z9 BC using a VSWITCH

Cipher	Without dynamic engine				With dynamic engine ibmca				
	real [s]	user [s]	sys [s]	tp [MB/s]	real [s]	user [s]	sys [s]	tp [MB/s]	encryption via
default	33.6	12.2	1.8	6.3	13.0	2.4	2.2	18.2	CPACF
3des-cbc	70.0	31.1	2.3	3.0	12.6	2.6	2.1	18.2	CPACF
aes128-cbc	33.0	12.1	1.7	6.5	12.5	2.4	2.2	18.2	CPACF
aes192-cbc	36.6	14.0	1.7	5.7	38.2	14.5	1.8	5.6	sw in libica

using an OSA Express2 SX adapter. In Table 5 we can see the positive impact of the CPACF on user time (on the client side). It is evident that for AES-192 there is no improvement, as this cipher is not supported by the CPACF of an IBM System z9 BC.

Out of curiosity, we repeat this test, but instead of using an OSA Express2 adapter, we now connect the two Linux guests using a VSWITCH. The results of these tests are collected in Table 6. The results are comparable. Perhaps there is an indication that using VSWITCH slightly increases the system time. Note that for all above tests, we are using in total only one CPU (IFL) for the z/VM system and its Linux guests.

When we change the Linux Version (SLES 10 SP2, SLES 10 SP3, or SLES 11 GA) used in the above tests, we do not see any significant difference.

8.4 Crypto Express support for RSA with SSH

We verify only that RSA hardware cryptographic support is also available for OpenSSH, and do not spend much effort studying the effect in terms of performance and throughput. For OpenSSH, we expect a greater benefit from CPACF than from the Crypto Express feature. Compared to common Web scenarios, the relationship between RSA handshakes and encrypted data transmission is different for SSH sessions. Usually there is only the RSA handshake at the beginning of a long session with high data transfer volumes.

As a test environment, we use a Linux server with SUSE SLES 10 SP3 on am IBM System z9 with two CPUs.

```
tmcc-123-168:~ # cat /proc/cpuinfo
vendor_id       : IBM/S390
# processors    : 2
bogomips per cpu: 2293.76
features        : esan3 zarch stfle msa ldisp eimm dfp
processor 0: version = FF,  identification = 01A75E,  machine = 2094
processor 1: version = FF,  identification = 01A75E,  machine = 2094
```

```
tmcc-123-168:~ #

tmcc-123-168:~ # uname -a
Linux tmcc-123-168 2.6.16.60-0.54.5-default #1 SMP Fri Sep 4 01:28:03 UTC 2009
s390x s390x s390x GNU/Linux

gnirss@tmcc-123-168:~> cat /etc/SuSE-release
SUSE Linux Enterprise Server 10 (s390x)
VERSION = 10
PATCHLEVEL = 3
```

Our test server has already one active CEX2C card, which is dedicated to this Linux guest[6].

```
tmcc-123-168:~ # vmcp q crypto
AP 01 CEX2C Queue 02 dedicated
```

The z90crypt driver is already loaded

```
tmcc-123-168:~ # rcz90crypt status
Checking for module z90crypt:                                    running
```

and dynamic engine loading support for OpenSSL is already active.

The preliminary package for SLES 10 is stored in the directory */root* and we install it on this server using *rpm -Uvh*

```
tmcc-123-168:~ # rpm -Uvh /root/openssh-4.2p1-18.40.35.828.0.PTF.552126.s390x.rpm
Preparing...                ########################################### [100%]
   1:openssh                ########################################### [100%]
Updating etc/sysconfig/ssh...
Starting SuSEconfig, the SuSE Configuration Tool...
Running module permissions only
Reading /etc/sysconfig and updating the system...
Executing /sbin/conf.d/SuSEconfig.permissions...
Finished.
```

and we verify the state of the updated package

```
tmcc-123-168:~ # rpm -qa | grep openssh
openssh-askpass-4.2p1-18.40.35
openssh-4.2p1-18.40.35.828.0.PTF.552126
```

We do not restart the SSHD server at this time.

Before we open a new SSH session, we check the current number of already completed RSA cryptographic requests on our Crypto Express card. This can be done either by checking the information in the *proc* file system, or with the newer interface in the *sysfs* file system. In Example 24, we see that 0x510 (this number is in hex) requests have been executed on the CEX2C card. The corresponding information retrieved from *sysfs* file system is shown in Example 25, where we see that on our *card01* already 1296 (this number is in decimal) requests have been executed since the device driver z90crypt was loaded.

```
tmcc-123-168:~ # cat /proc/driver/z90crypt

zcrypt version: 2.1.1
Cryptographic domain: 2
Total device count: 1
PCICA count: 0
PCICC count: 0
PCIXCC MCL2 count: 0
PCIXCC MCL3 count: 0
```

[6] For this scenario a Crypto Express configured as accelerator (CEX2A) and accessed in shared mode (CRYPTO APVIRT) is sufficient.


```
CEX2C count: 1
CEX2A count: 0
requestq count: 0
pendingq count: 0
Total open handles: 2


Online devices: 1=PCICA 2=PCICC 3=PCIXCC(MCL2) 4=PCIXCC(MCL3) 5=CEX2C 6=CEX2A
           0500000000000000 0000000000000000 0000000000000000 0000000000000000


Waiting work element counts
           0000000000000000 0000000000000000 0000000000000000 0000000000000000


Per-device successfully completed request counts
    00000000 00000510 00000000 00000000 00000000 00000000 00000000 00000000
    00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
    00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
    00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
    00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
    00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
    00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
    00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
```

Example 24: Query number of completed RSA requests using proc file system

```
tmcc-123-168:~ # cat  /sys/bus/ap/devices/card01/request_count
1296
```

Example 25: Query number of executed RSA requests using sysfs file system

Now we open a new SSH session from the PC to the Linux host:

```
gnirss@gnirss-tmcc:~$ ssh gnirss@9.152.123.168
Password:
Last login: Mon Jan 11 18:44:20 2010 from dyn-9-152-226-38.boeblingen.de.ibm.com
gnirss@tmcc-123-168:~>
```

After the new session is open, we verify whether the number of RSA requests has increased. We see (Example 26) that the counter has increased by 3 to 1299. Using the *proc* filesystem we would also see an increase by 3 to 0x513.

```
tmcc-123-168:~ # cat  /sys/bus/ap/devices/card01/request_count
1299
```

Example 26: Query number of completed RSA requests using sysfs file system - counter increased

In a second step we compare the behavior when we use the *scp* command. When we download data from the host to the PC using our usual network we see that also for an SCP session, the counter of completed RSA requests on the CEX2C card is increased by 3 request for each session.

```
gnirss@gnirss-tmcc:~/MG_MSC/sshCPACF$ time scp gnirss@9.152.123.168:testdata.txt .
Password:
testdata.txt                                  100%  196MB  11.5MB/s   00:17

real    0m20.389s
user    0m6.668s
sys     0m2.420s
```

The command *top* on the host shows that we have a CPU load of approximately 25% for *sshd* server (see Example 27).

```
top - 19:04:39 up  6:02,  4 users,  load average: 0.34, 0.21, 0.07
Tasks:  62 total,   3 running,  59 sleeping,   0 stopped,   0 zombie
Cpu(s):  7.2%us,  6.8%sy,  0.0%ni, 67.2%id, 13.3%wa,  1.8%hi,  2.2%si,  1.5%st
Mem:   1020144k total,   781608k used,   238536k free,    19388k buffers
Swap:        0k total,        0k used,        0k free,   690112k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 3553 gnirss    16   0  9784 2352 1304 R   25  0.2   0:03.31 sshd
 3554 gnirss    16   0  5164 1576 1244 S    4  0.2   0:00.57 scp
   76 root      15   0     0    0    0 D    1  0.0   0:00.79 pdflush
--- some lines not displayed ---
```

Example 27: Output of top command while SCP download file from host to PC

When we repeat the same test, but having dynamic engine loading disabled, we see a higher CPU load on the host of approximately 79% for *sshd* server (see Example 28), but as expected no significant increase in the elapsed time for the data transfer (compare with chapter 8.2 and remember, that the numbers here can only be a rough indication).

```
top - 19:07:13 up  6:04,  4 users,  load average: 0.38, 0.24, 0.10
Tasks:  62 total,   4 running,  58 sleeping,   0 stopped,   0 zombie
Cpu(s): 33.9%us,  4.5%sy,  0.0%ni, 35.3%id, 17.8%wa,  2.3%hi,  2.0%si,  4.2%st
Mem:   1020144k total,   784108k used,   236036k free,    19640k buffers
Swap:        0k total,        0k used,        0k free,   692440k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 3631 gnirss    16   0  9512 2172 1280 R   79  0.2   0:05.77 sshd
 3632 gnirss    16   0  5164 1576 1244 R    3  0.2   0:00.32 scp
   76 root      15   0     0    0    0 D    1  0.0   0:01.10 pdflush
--- some lines not displayed ---
```

Example 28: Output of top command while SCP download file from host to PC using dynamic engine ibmca

Finally, we want to see, whether there is a CPU load reduction, when a Crypto Express card is used for SCP file transfer. For this purpose we use a very short file with 2 Bytes (see Example 29) and send it via SCP to *localhost*.

```
gnirss@tmcc-123-168:~> ls -al short.txt
-rw-r--r-- 1 gnirss users 2 2010-01-12 12:46 short.txt
```

Example 29: very short file for data transfer

When we have dynamic engine loading support active to use hardware cryptographic support with CPACF and CEX2C we see a user time for our test case of approximately 0.01sec (see Example 30).

```
gnirss@tmcc-123-168:~> time scp short.txt gnirss@localhost:/home/gnirss/test1.txt
Password:
short.txt                                     100%    2     0.0KB/s   00:00

real    0m2.046s
user    0m0.010s
sys     0m0.006s
```

Example 30: SCP file transfer of a very short file to localhost

In the above test case, CPU cycles for the RSA work are offloaded to the CEX2C card. To approximate the number of cycles that can be offloaded with one *scp* command, we simply set our crypto card off-line using the following command:

Table 7: SFTP to localhost on a IBM System z9 EC to /dev/null

Cipher	Without dynamic engine				With dynamic engine ibmca				
	real [s]	user [s]	sys [s]	tp [MB/s]	real [s]	user [s]	sys [s]	tp [MB/s]	encryption via
default	18.8	11.7	0.7	13.1	6.5	2.1	0.7	65.3	CPACF
3des-cbc	35.6	25.4	0.7	6.5	7.2	2.3	0.7	65.3	CPACF
aes128-cbc	16.8	11.6	0.7	13.1	6.5	2.1	0.7	65.3	CPACF
aes192-cbc	18.7	13.2	0.7s	12.3	18.4	12.9	0.7	12.3	sw in libica

```
tmcc-123-168:~ # echo 0 > /sys/bus/ap/devices/card01/online
```

Then we repeat the file transfer with the very short file. When the card is off-line, the user time increases in our test from approximately 0.01 seconds to approximately 0.02 seconds.

As expected, the amount of the user time that can be offloaded to the CEX2C card is limited. In our test case it is approximately 0.01 second for the RSA operations per handshake when the SCP/SSH session is opened. For OpenSSH, the Crypto Express feature would show its advantage where large numbers of users (sessions) were performing a login with SSH at the same time.

8.5 Using SFTP

SSH File Transfer Protocol (or Secure File Transfer Protocol, SFTP) is an extension of SSH that provides file access, file transfer and file management functions. We test also SFTP (see Example 31) to confirm the observations of our experiences with SCP. The result is summarized in Table 7 and shows a similar reduction in user time (reduction in CPU cycles) if the cipher can be executed with the help of CPACF.

```
gnirss@tmcc-123-180:~/test11122009> time sftp -o ciphers=aes128-cbc
gnirss@localhost:test11122009/testdata.txt /dev/null
Connecting to localhost...
Password:
Fetching /home/gnirss/test11122009/testdata.txt to /dev/null
/home/gnirss/test11122009/testdata.txt          100%  196MB  65.3MB/s   00:03

real    0m6.518s
user    0m2.075s
sys     0m0.674s
```

Example 31: SFTP file transfer to localhost

8.6 rsync over SSH

The tool *rsync* can be used for either copying files locally on the current host, or for copying to or from a remote host. For remote copying, rsync uses SSH as a default for its communication. Because of this, rsync benefits automatically from hardware cryptographic support if SSH can use hardware cryptographic support. We can successfully verify this by enabling and disabling dynamic engine loading support for ibmca and compare the user time on the system for the execution of the rsync. The results are similar to those presented for SCP in the chapters 8.1, 8.2 and 8.3. This is as expected, because SSH is used under the covers.

8.7 Conclusion

The support of CPACF for SSH-related workload reduces the user time dramatically. This means that the load for a given SSH workload is considerably reduced on the involved CPUs. Usually this will lead to better performance for the end user if no other bottlenecks are present within the infrastructure (for example network limitations or disk I/O). In our environment, and for the given SSH scenarios, using the Crypto Express feature does not contribute much to CPU cycle reduction.

The improvement factor from CPACF for SSH-related workload is smaller than we saw for the *openssl speed* test program (see Table 1). Whereas the *openssl speed* test program deals only with encryption of data, SSH does a lot more work, such as reading data from disk, transferring data using the TCP/IP stack, and so forth. Therefore, the pure encryption part for the SSH workload is a lot smaller. CPACF only acts on this smaller part of the work and therefore the factor of improvement cannot reach the value observed with the pure encryption test utility. But there is still a considerable improvement.

9 Select cipher suite and MAC

Not all ciphers and message authentication code (MAC) algorithms are supported by CPACF. To benefit from IBM System z CPACF support, an appropriate cipher and MAC for encryption and data integrity protection should be selected when a SSH session is established. The SSH client and SSHD server negotiate which cipher and which MAC will be used during the session. Both client and server have a list of ciphers and MACs with a default search order. The list and the default search order can be adapted depending on your needs. If you want to benefit from CPACF capability for the MAC, you should place SHA at the top of the default search order. If you have a IBM System z10, you might not want to place AES-192 or AES-256 at the end of the search order for available ciphers.

The default search order is important for all the cases where a cipher is not explicitly specified when the user issues an *ssh* command. We assume that this will be the norm, and explicitly specifying a cipher (see Example 19) or explicitly specifying a MAC (see Example 36) is an exception.

9.1 SSHD server configuration

To determine which algorithms can be used by the SSHD server and to determine their search order, the configuration file */etc/ssh/sshd_config* of the SSH server can be modified.

To specify the ciphers permitted and the search order, the keyword *Ciphers* (for protocol version 2) can be used in the configuration file. Multiple ciphers must be comma-separated. The default order is

```
aes128-cbc,3des-cbc,blowfish-cbc,cast128-cbc,arcfour,
          aes192-cbc,aes256-cbc,aes128-ctr,aes192-ctr,aes256-ctr
```

If you are using an IBM System z9, this default order for the ciphers is good, as AES-128 and TDES are at the top.

To specify the message authentication code algorithms permitted, which are used for data integrity protection, and the relevant search order, the keyword *MACs* (for protocol version 2) can be used in the configuration file. Multiple algorithms must be comma-separated. The default order is

```
hmac-md5,hmac-sha1,hmac-ripemd160,hmac-sha1-96,hmac-md5-96
```

To test the effect of the *Ciphers* keyword in the SSHD configuration, we modify the *sshd_config* file to contain only TDES (see Example 32). We can verify that the default for a SCP test is now TDES, and if AES is specified explicitly in a *scp* command (see Example 33) an error message is issued. Please note, that a modification of the *sshd_config* file only takes effect after a restart of the SSHD daemon.

```
--- some lines not displayed ---
# Disable legacy (protocol version 1) support in the server for new
# installations. In future the default will change to require explicit
# activation of protocol 1
```



```
Protocol 2
Ciphers 3des-cbc
--- some lines not displayed ---
```

Example 32: sshd_config *file: modification to permit only TDES for test*

```
gnirss@tmcc-123-180:~/test11122009> scp -c aes128-cbc
testdata.txt gnirss@localhost:/dev/null
no matching cipher found: client aes128-cbc server 3des-cbc
lost connection
```

Example 33: Test with SCP and specifying a cipher not supported by the sshd_config *file*

9.2 SSH client configuration

To determine which algorithms can be used by the SSH client and their search order, the configuration file */etc/ssh/ssh_config* of the SSH client (see Example 34) can be modified.

```
--- some lines not displayed ---
#   Port 22
    Protocol 2
#   Cipher 3des
#   Ciphers aes128-cbc,3des-cbc,blowfish-cbc,cast128-cbc,arcfour,aes192-cbc,
#           aes256-cbc
#   MACs hmac-md5,hmac-sha1,umac-64@OpenSSH.com,hmac-ripemd160
--- some lines not displayed ---
```

Example 34: default ssh_config *file*

You might consider using the *MACs* keyword to put SHA algorithm at the top of the search order for message authentication codes. On an IBM System z10, you might consider using the *Ciphers* keyword to have the AES-192 and AES-256 algorithm not at the end of the search order.

9.3 Sample: Comparison between SHA-1 with CPACF support and MD5 in software

OpenSSH uses Hash-based Message Authentication Codes (HMAC). Today, Cipher-based MACs (CMAC) using TDES or AES are currently not implemented in OpenSSH. Therefore in what follows we deal only with HMAC based on MD5 or SHA[7]. In the default order of the message authentication codes for the SSH client as well as for the SSHD server, the algorithm MD5 (*hmac-md5*) is in the first position, whereas SHA-1 (*hmac-sha1*) is in the second position. The algorithm MD5 will always be executed in software on an IBM System z. The SHA algorithm can be executed inside of the CPACF.

Independent from the selection of a MAC for protection of user data integrity, there are some hashing operations during OpenSSH session negotiation. Mainly there are some SHA-1 and SHA-256 operations necessary for the key exchange (in case of having dynamic engine loading enabled, these operations are also executed in the CPACF). These additional hashing requests do not have an impact on the effects described below.

In the following tests we compare the effect of selecting SHA-1 combined with CPACF support, against the selection of MD5.

[7] Actually SHA-1 can be considered weak in comparison with SHA-256 or SHA-512, but the RFC 4453 for the SSH Transport Layer Protocol defines only MACs based on MD5 and SHA-1 for protecting data integrity. Just for the more critical key exchange RFC 4419 defines SHA-256.

Table 8: Influence of MAC: SCP file transfer with AES-128 on a IBM System z9 EC

Used MAC	User time [s]	Throughput [MB/s]
MD5	1.9	39.2
SHA-1	1.2	65.3

9.3.1 Prefer SHA-1 by adapting the SSH configuration

We adapt our configuration files for SSH and SSHD to check the effect of using SHA-1 with CPACF support for a file transfer with SCP. We are again using the scenario of Chapter 8.1. With *hmac-sha1* at the first position of the MACs in the configuration files we get about 1.2 seconds for the user time (see Example 35) compared to about 1.9 seconds when the default order is used on an IBM System z9 EC. The resulting output of the *scp* command shows also an increase in throughput from 39.2 MB per second to 65.3 MB per second. Table 8 contains the comparison for the *aes-128-cbc* algorithm between MD5 (in software) and SHA-1 (using CPACF) on a System z9 EC.

```
gnirss@tmcc-123-180:~/test11122009> time scp -c aes128-cbc
testdata.txt gnirss@localhost:/dev/null
Password:
testdata.txt                  100%  196MB  65.3MB/s   00:03

real    0m6.257s
user    0m1.192s
sys     0m0.803s
```

Example 35: SCP to local host example with AES-128

In our scenario with SCP on an IBM System z9 EC using the SHA-1 algorithm supported with CPACF, there is a clear advantage compared to MD5 (see summary in Table 8). The user time is reduced by more than one third.

For compatibility reasons, you might want to keep the MD5 algorithm in the search order, but put the SHA algorithm in the first position.

9.3.2 Specify MAC explicitly with the *scp* command

It is possible to specify not only the cipher but also the MAC when an *scp* command is issued. We test the *scp* command with MAC explicitly specified (see Example 36) on an IBM System z10 EC and compare the effect of the selected MAC.

```
h4213003:~ # time scp  -c aes128-cbc -o MACs=hmac-sha1
testdata.txt root@localhost:/dev/null
Password:
testdata.txt              100%  196MB 196.0MB/s   00:01

real    0m4.749s
user    0m0.597s
sys     0m0.499s
```

Example 36: Sample: SCP with cipher AES-128 and MAC SHA-1 on a z10 EC

Table 9 contains the results for using SHA-1 as MAC.

When we compare the values for the user time with dynamic engine ibmca with those of Table 3, we see a reduction in user time (CPU cycle consumption), of more than 30% for the AES algorithms when SHA-1 is used.

Table 9: SCP to localhost on a IBM System z10 EC with MAC=SHA-1

Cipher	Without dynamic engine				With dynamic engine ibmca			
	real [s]	user [s]	sys [s]	tp [MB/s]	real [s]	user [s]	sys [s]	tp [MB/s]
default	10.4	5.9	0.5	28.0	5.1	0.6	0.4	196.0
3des-cbc	20.7	15.7	0.6	11.5	6.0	1.0	0.4	98.0
aes128-cbc	10.5	5.8	0.8	28.0	4.7	0.6	0.5	196.0
aes192-cbc	12.2	6.5	0.5	24.5	5.4	0.6	0.5	98.0

Table 10: SCP between Linux guests inside of one z/VM using OSA Express2 SX: Influence of MAC on a IBM System z9 BC

Cipher/MAC	With dynamic engine ibmca			
	real [s]	user [s]	sys [s]	tp [MB/s]
3des-cbc/MD5	11.4	2.7	1.3	20.0
3des-cbc/SHA-1	9.5	1.8	0.9	25.0
aes128-cbc/MD5	10.4	2.4	1.2	22.2
aes128-cbc/SHA-1	9.3	1.5	1.0	28.6

In addition to the above test, we repeat the scenario of Chapter 8.3, and copy data from one Linux guest to another Linux guest running on a system z9 BC as shown in Example 37. Here we also observe a user time reduction (see Table 10) of more than 30% when SHA-1 with CPACF support can be used compared to MD5 as MAC.

```
zvmgast7:~ # time scp -c aes128-cbc -o MACs=hmac-sha1
/tmp/meter 10.3.3.23:/tmp/meter2
Password:
meter                         100%  200MB  28.6MB/s   00:07

real    0m9.284s
user    0m1.482s
sys     0m0.965s
```

Example 37: Sample: SCP with cipher AES-128 and MAC SHA-1 on a IBM System z9 BC between Linux guests inside of one z/VM

9.3.3 Specify MAC explicitly with the *sftp* command

It is also possible to specify the desired MAC with the *sftp* command (see Example 38). We compare for the *sftp* command the effect of the chosen MAC. In Table 11, the results are summarized and these also show a clear impact due to CPACF usage.

```
h4213003:~ # time sftp -o ciphers=aes128-cbc -o MACs=hmac-sha1
root@localhost:/root/testdata.txt /dev/null
Connecting to localhost...
Password:
Fetching /root/testdata.txt to /dev/null
/root/testdata.txt                            100%  196MB 196.0MB/s   00:01
```

Table 11: SFTP from localhost on a IBM System z10 EC MAC=SHA-1 and MD5

Cipher/MAC	Without dynamic engine				With dynamic engine ibmca			
	real [s]	user [s]	sys [s]	tp [MB/s]	real [s]	user [s]	sys [s]	tp [MB/s]
3des-cbc/MD5	19.2	15.4	0.4	12.3	5.8	1.5	0.4	65.3
3des-cbc/SHA-1	21.2	15.9	0.3	11.5	4.7	1.2	0.4	98.0
aes128-cbc/MD5	9.3	5.4	0.4	32.7	5.6	1.0	0.3	98.0
aes128-cbc/SHA-1	9.6	6.0	0.3	28.0	4.4	0.7	0.3	196.0
aes192-cbc/MD5	11.2	6.1	0.3	32.7	5.9	1.0	0.3	98.0
aes192-cbc/SHA-1	10.0	6.7	0.3	28.0	4.7	0.7	0.3	196.0

```
real    0m4.390s
user    0m0.689s
sys     0m0.343s
```

Example 38: Sample: SFTP with cipher AES-128 and MAC SHA-1 on a IBM System z10 EC

9.3.4 Conclusion

Even though it is possible to specify explicitly the MAC in the *scp* command as shown in Example 36, or in the *sftp* command as shown in Example 38, we assume that most users of SSH or SCP will not specify explicitly the cipher and the MAC. Therefore, we recommend using the configuration file of the SSH client and SSHD server to indicate your preferred cipher and MAC, depending on the hardware capability of your IBM System z and guidelines of your company.

We can observe with our tests for SCP as well as for SFTP that when all the cryptographic operations are executed in software, MD5 tends to result in a slightly smaller user time than SHA-1. As soon as hardware cryptographic support can be used for SHA-1, the calculation of the MAC for SHA-1 results in a clear advantage for SHA-1. When SHA-1 can benefit from CPACF, there is a reduction in the user time.

10 Summary

Using hardware encryption support in combination with OpenSSH can save a lot of CPU cycles, which can lead to better performance or increases in throughput. At a minimum, you will benefit from a CPU load decrease for this type of workload. In our environments we see a reduction in user time (see Figure 3) for TDES by up to a factor of 15 on System z10 EC (a factor of 11 on IBM System z9 EC) and for AES by up to a factor of 9 on IBM System z10 EC (a factor of 5 on System z9 EC). Selecting SHA as MAC also contributes to the improvement.

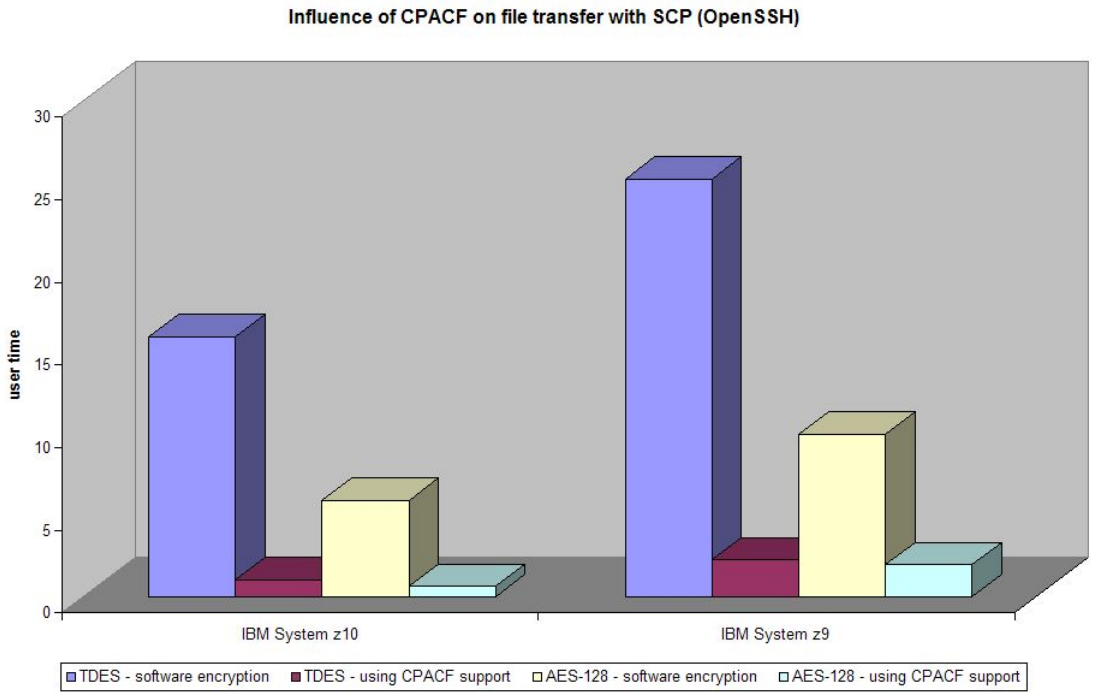

Figure 3: Benefit of using CPACF for encryption workload with SCP (OpenSSH)

Since using CPACF is free of charge and no additional hardware costs are incurred, we encourage you to exploit the CPACF feature. If LIC feature 3863 is not yet available on your IBM System z, we recommend installing it, as not only OpenSSH, but also other workloads can benefit.

If a Crypto Express feature is available on your IBM System z, please check whether you can access it with your Linux system as well. If not, you may want to adapt your *Activation Profile* settings or the z/VM directory to get access to the Crypto Express feature.

We recommend editing the SSH and SSHD config files to have TDES and AES at the top of the search order for ciphers, and SHA at the top of the search order for MACs. If you are using an IBM System z9 rather than an IBM System z10, consider editing the SSH and SSHD config files to have TDES or AES-128 at the top of the search order for ciphers.

As soon as the OpenSSH package supporting hardware encryption on IBM System z becomes available officially, you should install it to take advantage of the positive effects described in this paper.

The team who wrote this paper

This paper was produced during a workshop, and subsequent experiments and implementation, at the Technical Marketing Competence Center in the IBM Laboratory in Boeblingen, Germany.

Manfred Gnirss is a Senior IT specialist at the IBM Technical Marketing Competence Center (TMCC) and the Linux Center of Competence, Boeblingen, Germany. He holds a PhD in theoretical physics from the University of Tuebingen, Germany. Before joining the TMCC in 2000 he worked in z/VM® and z/OS® development for more than 12 years. Currently he is involved in several Linux for System z Proof-of-Concept projects and customer projects running at the TMCC.

Winfried Münch is an IT specialist at the german manufactuer of payment solutions ABK-Systeme GmbH, Dreieich, Germany. He has more than 7 years experience in software developing on several hardware platforms and operating systems, including IBM System z, z/OS and Linux for System z. He worked for two years as a System Engineer for IBM System z in z/VM and Linux for System z integration projects for various clients at PROFI Engineering System AG, Darmstadt, Germany.

Klaus Werner is a Senior Development Engineer working in the processor firmware development in IBM Systems and Technology Group. He studied Communications Engineering at the University of Cooperative Education of Stuttgart and received his Diplom-Ingenieur degree in 1984, joining IBM that same year to work in the Bring Up and Test of CMOS Processors. He has worked on projects in a variety of areas including RAS, System Feature development and in processor firmware development groups for IBM 9221, CMOS G1 to G6 processors, z900, z990, and z9. In 1995 he joined the Crypto team working on Cryptographic support for System z.

Arthur Winterling is a Software test specialist working for more than 20 years in software test. Arthur started as a tester for z/VSE™, over the years testing in other areas such as z/OS and PC applications. He is currently a member of the test team for Linux on System z. He has been one of the leaders in the Boeblingen test community for years, teaching test fundamentals and he is now involved in the testing of cryptographic functionality and solutions for Linux on IBM System z.

Acknowledgement

Our very best acknowledgement for discussions and helpful hints belong to
Rajiv Andrade, Felix Beck, Uwe Denneler, Theresa Elmendorf, Thomas Hanicke, Klaus Heinrich Kiwi, Cliff Laking, Dorothea Matthaeus, Ulrich Mayer, Elisabeth Puritscher, Peter Spera, Ruben Straus, Arwed Tschoeke, Ramon Valle from IBM,
to Ulrich Buch from ABK-Systeme GmbH,
and to Sascha Wehnert from Novell.

Acronyms

3DES Triple DES

AES Advanced Encryption Standard

CMAC Cipher-based Message Authentication Code

CMOS Complementary Metal Oxide Semiconductor

CPACF Central Processor Assist for Cryptographic Functions

CPU Central Processing Unit

DES Data Encryption Standard

FTP File Transfer Protocol

GA General Availability

HMAC Hash-based Message Authentication Code

HMC Hardware Management Console

IFL Integrated Facility for Linux

LPAR Logical Partition

LIC Licensed Internal Code

MAC Message Authentication Code

MD5 Message-Digest algorithm 5

OSA Open System Adapter

PRNG Pseudo Random Number Generator

PTF Programming Temporary Fix

RAS Reliability, Availability, and Serviceability

RSA Rivest, Shamir and Adleman algorithm

SCP Secure Copy Protocol

SE Support Element

SFTP Secure File Transfer Protocol, or also SSH File Transfer protocol

SHA Secure Hash Algorithm

SLES SUSE Linux Enterprise Server

SP Service Pack

SSH Secure Shell

SSHD Secure Shell Daemon

SSL Secure Sockets Layer

TDES Triple DES

TMCC Technical Marketing Competence Center

References

[1] IBM System z10 Enterprise Class cryptography for highly secure transactions.
`http://www-03.ibm.com/systems/z/advantages/security/z10cryptography.html`

[2] Security on z/VM, SG24-7471, 2007

[3] IBM System z10 Enterprise Class Configuration Setup, SG24-7571, 2008

[4] System z10 Support Element Operations Guide Version 2.10.0, SC28-6868

[5] Hardware Management Console Operations Guide Version 2.10.0, SC28-6867

[6] IBM J. RES. & DEV. Vol 51 NO.1/2 January/March 2007: Cryptographic system enhancements for the IBM System z9

[7] IBM J. RES. & DEV. Vol 53 NO.1 IBM System z10 design for RAS

Trademarks