Hardware cryptographic support of IBM z Systems for OpenSSH in RHEL 7.2 and SLES 12 SP1

Uwe Denneler, Harald Freudenberger, Paul Gallagher, Manfred Gnirss, Guillaume Hoareau, Arwed Tschoeke, Ingo Tuchscherer, Arthur Winterling

August 18, 2016

Abstract

This article summarizes our experiences with the configuration and usage of OpenSSH using hardware cryptographic support of IBM z Systems. We report our findings in the areas of performance and throughput improvement. Our positive experience indicates that you should make use of this capability when using OpenSSH.

Contents

1 Introduction

Access methods or protocols for Linux servers such as telnet, or FTP, should not only be avoided in Internet environments but also in internal company networks. This is because sensitive information such as passwords are transferred in clear text over the network. Using SSH and SCP increases the security because the information sent using the network is encrypted. Not only passwords, but also data is protected by encryption technologies. By nature, encryption of data is expensive and can heavily impact performance, throughput, or CPU load of a system. IBM® z Systems® 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 z Systems® cryptographic hardware support, if a specific flag (*−−with−ssl−engine*) is used during the build of the OpenSSH package. This support has been available for six years, therefore we describe our current experiences using hardware accelerated encryption for OpenSSH and how setup and configuration have been simplified in this area, as well as our findings about performance and throughput improvement over the last years (compare with (see also [1]).

For our tests, we used IBM z Systems z13™ and SUSE Linux Enterprise Server (SLES) 12 SP1 and Red Hat Enterprise Server RHEL 7.2.

2 Hardware cryptographic support of z Systems

IBM z Systems 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 z Systems. It has been introduced with z990 and z890. The CPACF incorporated in IBM z13® delivers support for symmetric encryption algorithms Data Encryption Standard (DES), Triple DES (TDES), Advanced Encryption Standard (AES), hashing algorithm SHA and Pseudo Random Number Generator (PRNG). 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 z Systems z13, it is the Crypto Express5 feature (CEX5S). The Crypto Express feature can be configured as Accelerator (CEX5A), or as Coprocessor (CEX5C), or in EP11 mode (CEX5P). If the feature is configured as CEX5A, it can perform clear key RSA operations with very high speed. If configured as CEX5C, it can perform asymmetric operations (RSA) in clear key mode and also in secure key mode. Note that the operations executed by the Crypto Express feature are performed asynchronously outside of the central processor. This means, work is off-loaded and CPU cycles are reduced (i.e. less load on the CPU).

And last but not least, there is a hybrid way: With Protected Key operations the high performance for data encryption using the CPACF is used, while the privacy of the cryptographic key material is guaranteed by using the CEX5C.

To benefit from the CPACF, you must install LIC internal feature 3863 (Crypto Enablement feature), which is available free of charge (see also [2], [3]). By default, the IBM z Systems 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.

It is recommended to install the Crypto Enablement feature even if you do not intend to use the Crypto Express5 feature, because there is already a considerable benefit from an active CPACF.

2.1 Verification of installed LIC 3863 using the SE

You can check if the CPACF is enabled in your environment using the dialogues provided on the Support Element (SE). In the *System Details* panel you can find “CP Assist for Crypto functions: Installed” (see Figure 1), or “CP Assist for Crypto functions: Not installed”.

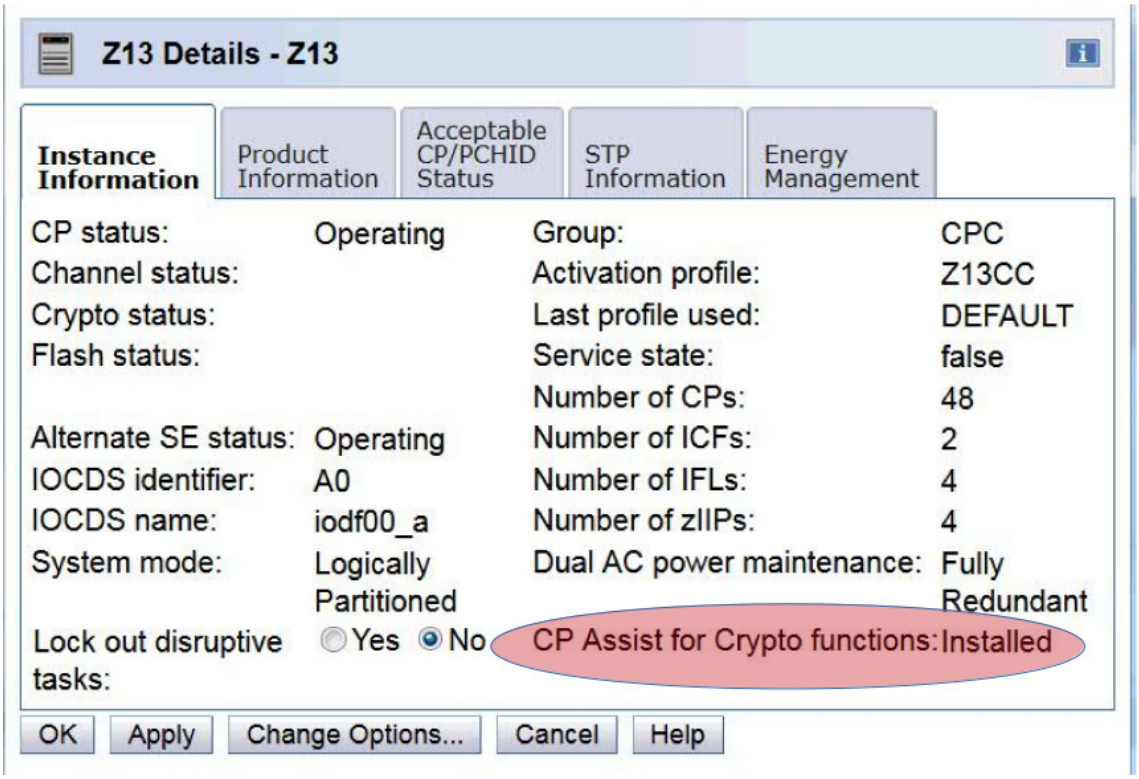

Figure 1: IBM z13: LIC 3863 is installed

2.2 Verification of installed LIC 3863 using a Linux command

A Linux for z Systems user can easily check 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 implementation inside the libica library. This command is available if the libica package is installed on the Linux for z Systems server.

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). For all other algorithms, you will find a *no* in column *# hardware* in the output of the *icainfo* command.

```
gnirss@zlx14020:~> icainfo
The following CP Assist for Cryptographic Function (CPACF)
operations are supported by libica on this system:
 function       | # hardware | #software
----------------+------------+-----------
        SHA-1   |     yes    |     yes
      SHA-224   |     yes    |     yes
      SHA-256   |     yes    |     yes
      SHA-384   |     yes    |     yes
      SHA-512   |     yes    |     yes
        P_RNG   |      no    |     yes
       RSA ME   |      no    |     yes
      RSA CRT   |      no    |     yes
      DES ECB   |      no    |     yes
      DES CBC   |      no    |     yes
*** some lines not displayed ***
```

Example 1: Response of icainfo, 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[1].

```
gnirss@zlx14020:~> icainfo
The following CP Assist for Cryptographic Function (CPACF)
operations are supported by libica on this system:
 function       | # hardware | # software
----------------+------------+------------
        SHA-1   |     yes    |     yes
      SHA-224   |     yes    |     yes
      SHA-256   |     yes    |     yes
      SHA-384   |     yes    |     yes
      SHA-512   |     yes    |     yes
        P_RNG   |     yes    |     yes
       RSA ME   |      no    |     yes
      RSA CRT   |      no    |     yes
      DES ECB   |     yes    |     yes
      DES CBC   |     yes    |     yes
   DES CBC CS   |     yes    |      no
      DES OFB   |     yes    |      no
      DES CFB   |     yes    |      no
      DES CTR   |     yes    |      no
   DES CTRLST   |     yes    |      no
  DES CBC MAC   |     yes    |      no
     DES CMAC   |     yes    |      no
     3DES ECB   |     yes    |     yes
     3DES CBC   |     yes    |     yes
  3DES CBC CS   |     yes    |      no
     3DES OFB   |     yes    |      no
     3DES CFB   |     yes    |      no
     3DES CTR   |     yes    |      no
 3DES CTRLIST   |     yes    |      no
 3DES CBC MAC   |     yes    |      no
    3DES CMAC   |     yes    |      no
      AES ECB   |     yes    |     yes
      AES CBC   |     yes    |     yes
   AES CBC CS   |     yes    |      no
      AES OFB   |     yes    |      no
      AES CFB   |     yes    |      no
      AES CTR   |     yes    |      no
   AES CTRLST   |     yes    |      no
  AES CBC MAC   |     yes    |      no
     AES CMAC   |     yes    |      no
      AES CCM   |     yes    |      no
      AES GCM   |     yes    |      no
      AES XTS   |     yes    |      no
```

Example 2: Encryption algorithms supported in CPACF of IBM z Systems z13

If you find a *no* in column *# software* in the output of the *icainfo* command (see Example 2), there is no software fallback implemented in libica (see also chapter 6 in [4]).

[1]The *no* for RSA ME and RSA CRT support in the column *# hardware* of Example 2 indicates that there is no access from the Linux server to a Crypto Express feature, or that the crypto device driver is not loaded.

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

If you have a Crypto Express5 (CEX5S) adapter in your z Systems, 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, see chapter 10 of [5] and chapter 6 of [6]. For details how to enable access to the CEX feature for a Linux system running in a z/VM® environment, see chapter 6 of [7] and [8]. In [9], information about how to work with the HMC can be found.

4 HW- Support - Architecture for OpenSSH

Here is an 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 $--with-ssl-engine$, the OpenSSL library can use available engines and load them dynamically.

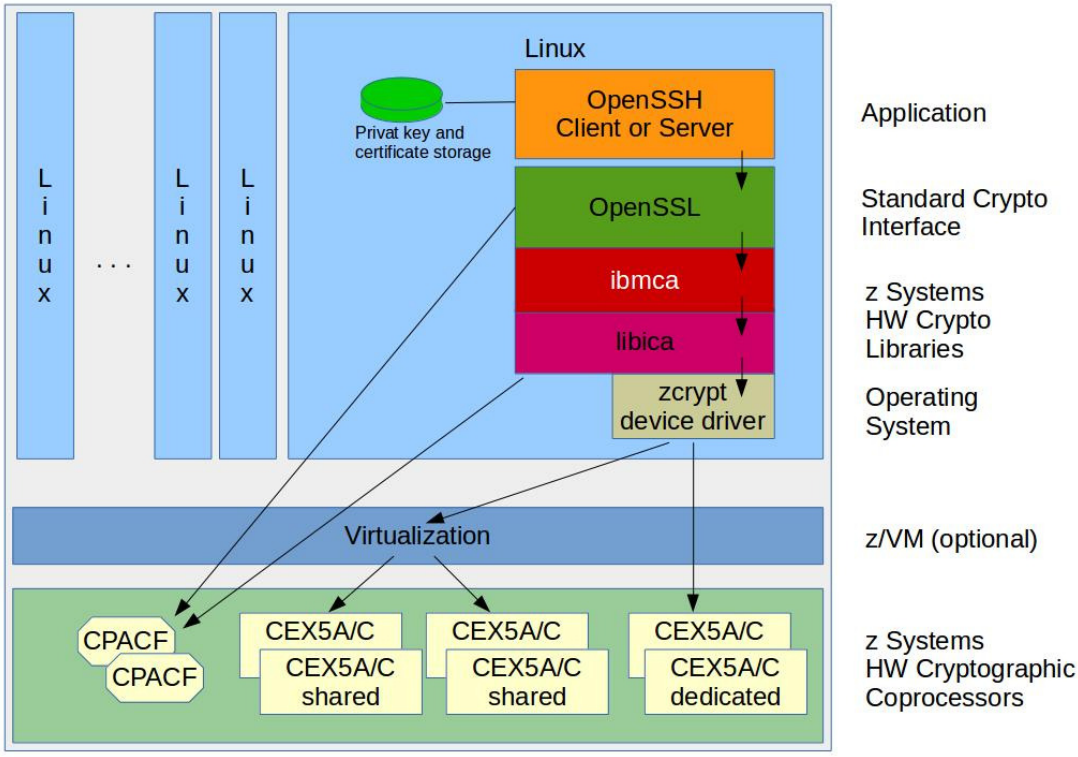

Figure 2: Linux for z Systems environment for hardware cryptographic support for OpenSSH

In a z Systems 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 via the underlying hardware CPACF or Crypto Express feature (if installed and available). 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 fallback[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 dedicate or 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 [7]).

If OpenSSL is not configured to use the ibmca engine, all cryptographic operations will be executed inside of OpenSSL. The most recent releases of OpenSSL provide built-in support for some crypto algorithms to be executed directly using CPACF instructions, providing LIC 3863 has been installed. Andy Polyakov has implemented the support for the AES and SHA algorithms in inline-assembler inside of OpenSSL. This means that even if the ibmca engine has not been installed or configured, as a minimum AES and SHA will execute faster due to the use of CPACF.

5 Our environment

For our test, we use Linux servers as virtual guests[3] in a z/VM LPAR of a IBM z13.

The following software and driver packages are needed on Linux for z Systems to enable OpenSSH to benefit from the complete hardware cryptographic support of IBM z Systems.

- openssh
- openssl
- openssl-ibmca
- libica
- zcrypt driver (device driver is part of system)

All these packages are part of the Linux for z Systems distributions. Depending on the distribution and installation parameters, some or all of them might be already installed with your initial set up.

```
gnirss@zlx14020:~> lscpu
Architektur:           s390x
CPU op-mode(s):        32-bit, 64-bit
Byte-Reihenfolge:      Big Endian
CPU(s):                2
On-line CPU(s) list:   0,1
Thread(s) pro Kern:    1
Kern(e) pro Socket:    1
Socket(s) per book:    1
Book(s):               2
Anbieterkennung:       IBM/S390
BogoMIPS:              20325.00
Hypervisor:            z/VM 6.3.0
Hypervisor-Anbieter:   IBM
Virtualisierungstyp:   voll
Dispatching-Modus:     horizontal
L1d Cache:             128K
L1i Cache:             96K
L2d Cache:             2048K
L2i Cache:             2048K
```

Example 3: Our environment - hardware server

[2]Starting with libica V2, libica uses the OpenSSL library for execution of cryptographic requests for some algorithms, if software fallback is necessary.

[3]The setup and configuration of Linux to use hardware cryptographic support is independent of whether the Linux is running natively in an LPAR, or as a guest in z/VM

We use two z/VM guests, one with SUSE SLES 12 SP1 and one with Red Hat RHEL 7.2 installed. The z/VM directory contains the CRYPTO statement to assign a dedicated crypto queue for each of our test Linux guests (see Example 4). For our first guest we use domain 5 and for the second we use domain 6.

```
USER ZLX14020 <password> 2G 4G G
. . . . . some lines not displayed . . . . .
* crypto
   CRYPT DOMAIN 5 APDED 0
. . . . . some lines not displayed . . . . .
USER ZLB14020 <password> 2G 4G G
. . . . . some lines not displayed . . . . .
* crypto
   CRYPT DOMAIN 6 APDED 0
. . . . . some lines not displayed . . . . .
```

Example 4: Extract of z/VM directory entry for Linux guests with dedicated access to CEX5S

Note that when using Crypto Express for OpenSSH, we could also use a virtualized crypto card for acceleration of the RSA handshake. Defining with CRYPTO APVIRT is sufficient for RSA (clear key) acceleration.

Also note that in the following we do not discuss any aspects of SELinux configuration.

5.1 Installation of SLES 12 SP1

We use a default installation of SUSE SLES 12 SP1. During installation, we specify installation with *System z HW crypto support* (see Figure 3).

The resulting software environment of our Linux server is shown below:

```
gnirss@zlx14020:~> uname -a
Linux zlx14020 3.12.49-11-default #1 SMP Wed Nov 11 20:52:43 UTC 2015 (8d714a0)
s390x s390x s390x GNU/Linux
```

Example 5: Our environment (SLES) - system and kernel

```
gnirss@zlx14020:~> cat /etc/os-release
NAME="SLES"
VERSION="12-SP1"
VERSION_ID="12.1"
PRETTY_NAME="SUSE Linux Enterprise Server 12 SP1"
ID="sles"
ANSI_COLOR="0;32"
CPE_NAME="cpe:/o:suse:sles:12:sp1"
```

Example 6: Our environment (SLES) - version/release of operating system

The following packages that are required for encryption, including hardware crypto support, are already installed:

```
gnirss@zlx14020:~> rpm -qa | grep openssl
libopenssl1_0_0 -1.0.1i-34.1.s390x
openssl-1.0.1i-34.1.s390x
openssl-ibmca-32bit -1.2.0-151.1.s390x
libopenssl1_0_0 -32bit -1.0.1i-34.1.s390x
openssl-ibmca-1.2.0-151.1.s390x

gnirss@zlx14020:~> rpm -qa | grep libica
libica2 -2.4.2-14.1.s390x
libica-2_3_0 -32bit -2.3.0-15.2.s390x
```



```
gnirss@zlx14020:~> rpm -qa | grep ibmca
openssl-ibmca-32bit-1.2.0-151.1.s390x
openssl-ibmca-1.2.0-151.1.s390x
```

Example 7: Our environment (SLES) - required packages are already installed

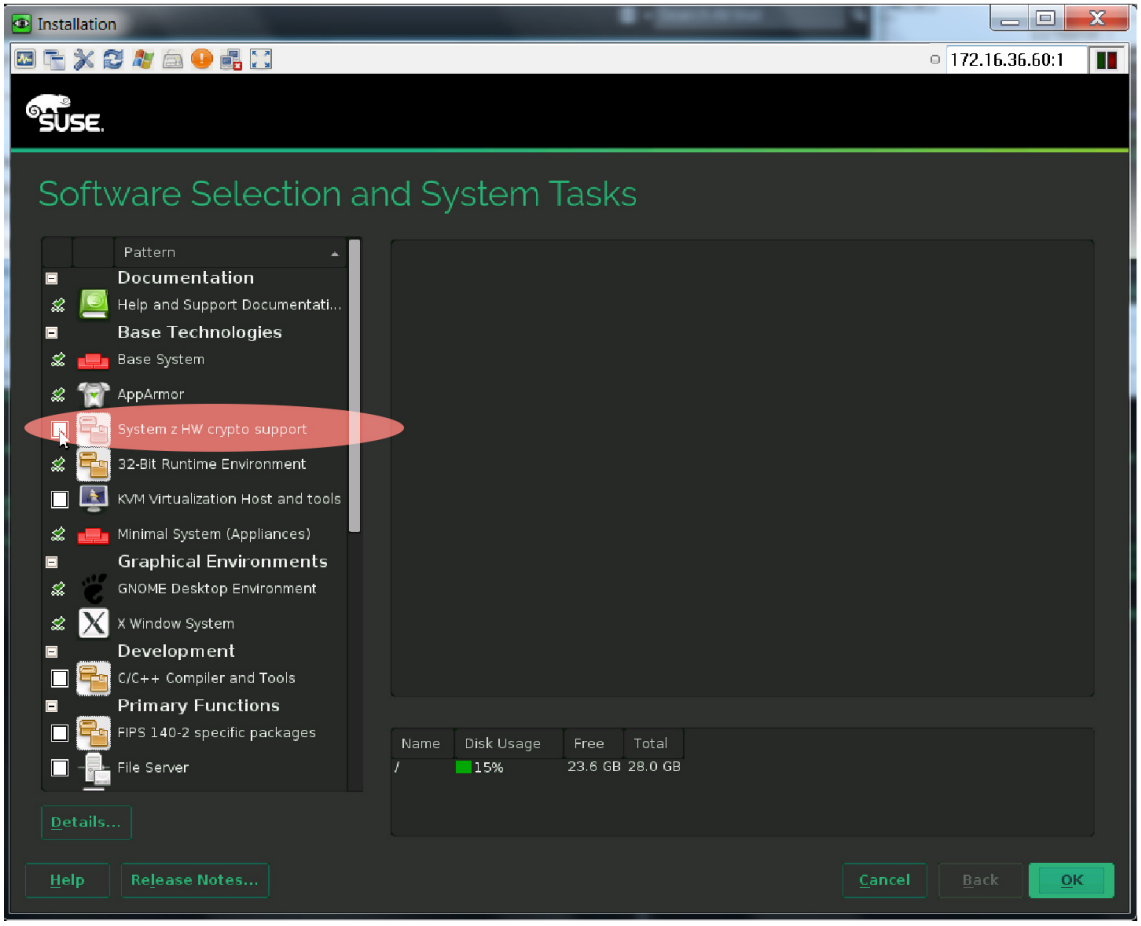

Figure 3: Installation of SLES 12 SP1 with HW crypto support

A first check also indicates that dynamic engine loading support is enabled by default and the engine ibmca is used in our installation

```
gnirss@zlx14020:~> openssl engine
(dynamic) Dynamic engine loading support
(ibmca) Ibmca hardware engine support
gnirss@zlx14020:~> openssl engine -c
(dynamic) Dynamic engine loading support
(ibmca) Ibmca hardware engine support
 [RSA, DSA, DH, RAND, DES-ECB, DES-CBC, DES-OFB, DES-CFB, DES-EDE3, DES-EDE3-CBC,
DES-EDE3-OFB, DES-EDE3-CFB, AES-128-ECB, AES-128-CBC, AES-128-CFB, AES-128-OFB,
AES-192-ECB, AES-192-CBC, AES-192-CFB, AES-192-OFB, AES-256-ECB, AES-256-CBC,
AES-256-CFB, AES-256-OFB, SHA1, SHA256]
```

Example 8: Dynamic engine support is enabled and ibmca engine is available

Now we check for the availability of the crypto queue by sending a command to the underlying z/VM,


```
gnirss@zlx14020:~> sudo vmcp q v crypto
AP 000 CEX5C Domain 005 dedicated
```

Example 9: Access to a crypto queue is available (Domain 5)

and we see that access is available as has been defined in the z/VM directory (see Example 4). Up to now we do not have the crypto device driver loaded (see Example 10), and therefore all RSA requests will be executed as software fallback in libica.

```
gnirss@zlx14020:~> sudo lszcrypt
gnirss's password:
lszcrypt: error - cryptographic device driver zcrypt is not loaded!
```

Example 10: Crypto device driver not loaded

Note that in order to use the vmcp and lszcrypt command, the package *s390-tools-1.24.1-49.4.s390x* has to be installed. For our z/VM guest, the z/VM privilege class G has been assigned (see also Example 4).

To load the crypto device driver, use the modprobe command

```
gnirss@zlx14020:~> sudo modprobe ap
```

Example 11: Load the crypto device driver

and verify whether it was successful (lsmod | grep ap). Ensure that the device driver will be re-loaded after a re-IPL (re-boot)[4] of the Linux server.

Now the lszcrypt command shows that access to the crypto device is available (see Example 12).

```
gnirss@zlx14020:~> sudo lszcrypt
card00: CEX5C
```

Example 12: Crypto device driver is loaded and accessible

Since the crypto device driver is now loaded, also indicated by the icastats command, the hardware support for RSA ME and RSA CRT is now available (see Example 13 and compare with Example 2).

```
gnirss@zlx14020:~> icainfo
The following CP Assist for Cryptographic Function (CPACF)
operations are supported by libica on this system:
 function      | # hardware | # software
---------------+------------+------------
        SHA-1  |     yes    |     yes
      SHA-224  |     yes    |     yes
      SHA-256  |     yes    |     yes
      SHA-384  |     yes    |     yes
      SHA-512  |     yes    |     yes
        P_RNG  |     yes    |     yes
       RSA ME  |     yes    |     yes
      RSA CRT  |     yes    |     yes
      DES ECB  |     yes    |     yes
      DES CBC  |     yes    |     yes
*** some lines not displayed ***
```

Example 13: RSA is available via hardware support

Remark: If you have installed SUSE SLES 12 SP1 without *System z HW crypto support* (see Figure 3), then you have to ensure that the required packages are installed manually, and you have to adapt the OpenSSL configuration file manually.

[4] SLES 12 SP1 - to load the crypto device driver at boot time, the config file */etc/modules-load.d/ap.conf* must be executable (use command *chmod +x ap.conf*) and must contain the name of the load module:
load module module at boot time
ap

5.2 Installation of RHEL 7.2

After the basic installation of Red Hat RHEL 7.2, including the *Security Tools* (see Figure 4),

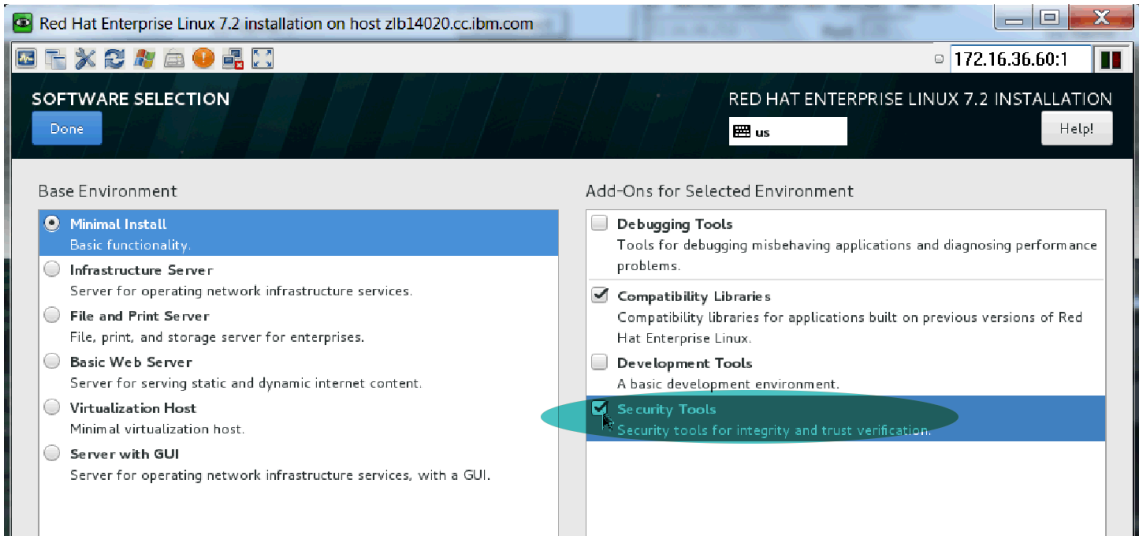

Figure 4: Installation of RHEL 7.2 with Security Tools

the packages for ibmca and libica have to be installed manually.

```
[gnirss@zlb14020 ~]$ rpm -qa | grep openssl
openssl-libs-1.0.1e-42.el7_1.9.s390x
openssl-ibmca-1.2.0-10.el7.s390x
openssl-1.0.1e-42.el7_1.9.s390x
openssl098e-0.9.8e-29.el7_0.2.s390x

[gnirss@zlb14020 ~]$ rpm -qa | grep libica
libica-2.4.2-1.el7.s390x

[gnirss@zlb14020 ~]$ rpm -qa | grep ibmca
openssl-ibmca-1.2.0-10.el7.s390x
```

Example 14: Our environment (RHEL) - required packages

The resulting software environment of our Linux server is shown below:

```
[gnirss@zlb14020 ~]$ uname -a
Linux zlb14020.cc.ibm.com 3.10.0-327.el7.s390x #1 SMP Thu Oct 29 17:32:48 EDT 2015
s390x s390x s390x GNU/Linux
```

Example 15: Our environment (RHEL) - system and kernel

```
[gnirss@zlb14020 ~]$ cat /etc/os-release
NAME="Red Hat Enterprise Linux Server"
VERSION="7.2 (Maipo)"

ID="rhel"
ID_LIKE="fedora"
VERSION_ID="7.2"
PRETTY_NAME="Red Hat Enterprise Linux Server 7.2 (Maipo)"

ANSI_COLOR="0;31"
```



```
CPE_NAME="cpe:/o:redhat:enterprise_linux:7.2:GA:server"
HOME_URL="https://www.redhat.com/"
BUG_REPORT_URL="https://bugzilla.redhat.com/"

REDHAT_BUGZILLA_PRODUCT="Red Hat Enterprise Linux 7"
REDHAT_BUGZILLA_PRODUCT_VERSION=7.2
REDHAT_SUPPORT_PRODUCT="Red Hat Enterprise Linux"
REDHAT_SUPPORT_PRODUCT_VERSION="7.2"
```

Example 16: Our environment (RHEL) - version/release of operating system

Now we check for the availability of the crypto queue by sending a command to the underlying z/VM

```
[gnirss@zlb14020 ~]$ sudo vmcp q v crypto
AP 000 CEX5C Domain 006 dedicated
```

Example 17: Access to a crypto queue is available (Domain 6)

and whether the device driver is loaded:

```
[gnirss@zlb14020 ~]$ lszcrypt
lszcrypt: error - cryptographic device driver zcrypt is not loaded!
```

Example 18: Crypto device driver is not loaded

As expected, the device driver for acceleration of RSA is not yet loaded. Therefore, a first check for the availability of the ibmca engine is negative (see Example 19).

```
[gnirss@zlb14020 ~]$ openssl engine
(dynamic) Dynamic engine loading support

[gnirss@zlb14020 ~]$ openssl engine -c
(dynamic) Dynamic engine loading support
```

Example 19: Engine ibmca is not yet available for OpenSSL

To make use of the ibmca engine to benefit from the implemented hardware support, you need to modify the configuration file of OpenSSL. 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. Ensure you take a backup of the configuration file before you change it.
2. Ensure that there is a ibmca_section at the end of the OpenSSL configuration file.
3. Insert the following line at the top of the configuration file.

   ```
   openssl_conf = openssl_def
   ```

   Ensure that this line appears only once in the config file. The configuration file looks now as shown in Example 20.
4. You might check the value of the dynamic_path variable and, if necessary, change it to the correct path.

```
[gnirss@zlb14020 ~]$ sudo vi /etc/pki/tls/openssl.cnf
sudo] password for gnirss:
#
# 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 inserted to enable dynamic engine ibmca *** MG ***
openssl_conf = openssl_def
# Extra OBJECT IDENTIFIER info:
#oid_file               = $ENV::HOME/.oid
oid_section             = new_oids

*** some lines not displayed ***

# 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 *** MG ***
#openssl_conf = openssl_def

[openssl_def]
engines = engine_section

[engine_section]

foo = ibmca_section

[ibmca_section]
dynamic_path = /lib64/openssl/engines/libibmca.so
engine_id = ibmca
init = 1
#
# The following ibmca algorithms will be enabled by these parameters
# to the default_algorithms line. Any combination of these is valid,
# with "ALL" denoting the same as all of them in a comma separated
# list.
#
# RSA
# - RSA encrypt, decrypt, sign and verify, key lengths 512-4096
#
# RAND
# - Hardware random number generation
#
# CIPHERS
# - DES-ECB, DES-CBC, DES-CFB, DES-OFB, DES-EDE3, DES-EDE3-CBC, DES-EDE3-CFB,
#   DES-EDE3-OFB, AES-128-ECB, AES-128-CBC, AES-128-CFB, AES-128-OFB,
#   AES-192-ECB, AES-192-CBC, AES-192-CFB, AES-192-OFB, AES-256-ECB,
#   AES-256-CBC, AES-256-CFB, AES-256-OFB symmetric crypto
#
# DIGESTS
# - SHA1, SHA256 digests
```



```
#
default_algorithms = ALL
#default_algorithms = RAND,RSA,CIPHERS,DIGESTS
```

Example 20: OpenSSL configuration file with dynamic engine loading support for ibmca

A first check now shows that the ibmca engine is available (see also Example 21).

```
[gnirss@zlb14020 ~]$ openssl engine
(dynamic) Dynamic engine loading support
(ibmca) Ibmca hardware engine support

[gnirss@zlb14020 ~]$ openssl engine -c
(dynamic) Dynamic engine loading support
(ibmca) Ibmca hardware engine support
 [RSA, DSA, DH, RAND, DES-ECB, DES-CBC, DES-OFB, DES-CFB, DES-EDE3, DES-EDE3-CBC,
DES-EDE3-OFB, DES-EDE3-CFB, AES-128-ECB, AES-128-CBC, AES-128-CFB, AES-128-OFB,
AES-192-ECB, AES-192-CBC, AES-192-CFB, AES-192-OFB, AES-256-ECB, AES-256-CBC,
AES-256-CFB, AES-256-OFB, SHA1, SHA256]
```

Example 21: Dynamic engine support for ibmca is enabled

As the crypto device driver is not yet loaded, the command icainfo still shows, that there is no hardware support in libica for RSA (see Example 22).

```
[gnirss@zlb14020 ~]$ icainfo
The following CP Assist for Cryptographic Function (CPACF)
operations are supported by libica on this system:
 function       | # hardware | # software
----------------+------------+------------
        SHA-1   |     yes    |     yes
      SHA-224   |     yes    |     yes
      SHA-256   |     yes    |     yes
      SHA-384   |     yes    |     yes
      SHA-512   |     yes    |     yes
        P_RNG   |     yes    |     yes
       RSA ME   |      no    |     yes
      RSA CRT   |      no    |     yes
      DES ECB   |     yes    |     yes
      DES CBC   |     yes    |     yes
*** some lines not displayed ***
```

Example 22: RSA support only as software fallback

This is consistent with the information available in sysfs: We see that up to now, there are no entries for the crypto card (ap) support (see Example 23):

```
[gnirss@zlb14020 ~]$ ls /sys/devices/ap/
ls: Zugriff auf /sys/devices/ap/ nicht möglich: Datei oder Verzeichnis
nicht gefunden

[gnirss@zlb14020 ~]$ ls /sys/devices/
css0  iucv  platform  qeth  scm  software  system  tracepoint  virtual
```

Example 23: sysfs without subdirectories for ap support

We have already verified that access to the crypto card is available (see Example 17). Therefore, we load now the crypto device driver

```
[gnirss@zlb14020 ~]$ sudo modprobe ap
```

Example 24: Loading crypto device driver ap

and verify whether it was successful (lsmod | grep ap). Ensure that the device driver will be re-loaded after a re-IPL (re-boot)[5] of the Linux server.

Now we can check again in sysfs for ap support.

```
[gnirss@zlb14020 ~]$ ls /sys/devices/
ap  css0  iucv  platform  qeth  scm  software  system  tracepoint  virtual

[gnirss@zlb14020 ~]$ ls /sys/devices/ap
card00  module  power  uevent

[gnirss@zlb14020 ~]$ ls /sys/devices/ap/card00/
ap_functions  hwtype     online          raw_hwtype      reset      uevent
depth         interrupt  pendingq_count  request_count   subsystem
driver        modalias   power           requestq_count  type
```

Example 25: sysfs with support for crypto card

We see that the crypto card is online ("1" in Example 26)

```
[gnirss@zlb14020 ~]$ cat  /sys/devices/ap/card00/online
1
```

Example 26: Crypto card is online

and that the card is a CEX5S ("11" in Example 27), which is configured in coprocessor mode ("CEX5C" in Example 27).

```
[gnirss@zlb14020 ~]$ cat /sys/devices/ap/card00/raw_hwtype
11

[gnirss@zlb14020 ~]$ cat /sys/devices/ap/card00/type
CEX5C
```

Example 27: Crypto Express5 card configured in coprocessor mode

Now we check for the number of executed requests in the crypto card (see Example 28). We will observe a change of this counter when we execute RSA requests using the crypto card.

```
[gnirss@zlb14020 ~]$ cat  /sys/devices/ap/card00/request_count
1
```

Example 28: Number of requests that are already processed by this device

Now we perform crypto operations which use the crypto card (i.e. RSA)

```
[gnirss@zlb14020 ~]$ openssl speed rsa2048 -elapsed
You have chosen to measure elapsed time instead of user CPU time.
Doing 2048 bit private rsa's for 10s: 6225 2048 bit private RSA's in 10.00s
Doing 2048 bit public rsa's for 10s: 6629 2048 bit public RSA's in 10.00s
OpenSSL 1.0.1e-fips 11 Feb 2013
built on: Tue Jun 23 11:14:09 EDT 2015
options:bn(64,64) md2(int) rc4(8x,char) des(idx,cisc,16,int) aes(partial)
idea(int) blowfish(idx)
compiler: gcc -fPIC -DOPENSSL_PIC -DZLIB -DOPENSSL_THREADS -D_REENTRANT
-DDSO_DLFCN -DHAVE_DLFCN_H -DKRB5_MIT -m64 -DB_ENDIAN -DTERMIO -Wall
-O2 -g -pipe -Wall -Wp,-D_FORTIFY_SOURCE=2 -fexceptions -fstack-protector-strong
--param=ssp-buffer-size=4 -grecord-gcc-switches   -m64 -march=z196 -mtune=zEC12
-Wa,--noexecstack -DPURIFY -DOPENSSL_BN_ASM_MONT -DOPENSSL_BN_ASM_GF2m
```

[5] RHEL 7.2 - to load the crypto device driver at boot time, the config file */etc/rc.modules* must be executable (use command *chmod +x /etc/rc.modules*) and must contain the command to load the module:
modprobe ap


```
-DSHA1_ASM -DSHA256_ASM -DSHA512_ASM -DAES_ASM -DAES_CTR_ASM -DAES_XTS_ASM
-DGHASH_ASM
                  sign    verify    sign/s verify/s
rsa 2048 bits 0.001606s 0.001509s    622.5    662.9
```

Example 29: Test for RSA requests

and we check for the counters after the test is completed. Now we see an increased number of requests (see Example 30). This means, the Crypto Express feature has been used.

```
[gnirss@zlb14020 ~]$ cat  /sys/devices/ap/card00/request_count
12859
```

Example 30: Number of requests that are processed by this device

Alternatively, we can verify whether RSA uses hardware crypto support via libica by using the icastats command (see Example 31).

```
[gnirss@zlb14020 ~]$ icastats
 function        |          # hardware          |         # software
-----------------+------------------------------+-----------------------------
                 |       ENC     CRYPT    DEC   |        ENC     CRYPT    DEC
-----------------+------------------------------+-----------------------------
          SHA-1  |               120            |                0
        SHA-224  |                 0            |                0
        SHA-256  |                 0            |                0
        SHA-384  |                 0            |                0
        SHA-512  |                 0            |                0
          P_RNG  |                 1            |                0
         RSA-ME  |              6375            |                0
        RSA-CRT  |              6512            |                0
*** some lines not displayed ***
```

Example 31: RSA requests performed with Hardware support in RHEL server

5.3 Configuring ibmca engine

In the ibmca section of the OpenSSL config file[6], it is possible to approximately determine the scope of the engine. You can either use the engine with its full capabilities (this is the default configuration), or you can include/exclude RSA, MACs, or the symmetric ciphers.

We mentioned already in chapter 4 that there is now a full SHA implementation included in OpenSSL which directly uses CPACF instructions. Therefore, we can exclude the calculation of SHA from ibmca. We modify the ibmca section from the default (as shown in Example 20) to exclude all DIGESTS (see Example 32).

```
*** some lines not displayed ***
# DIGESTS
# - SHA1, SHA256 digests
#
#default_algorithms = ALL
default_algorithms = RAND,RSA,CIPHERS
```

Example 32: ibmca setion in OpenSSL configuration file without DIGESTS

The possibility to exclude algorithms might also be of interest if there is no access to a Crypto Express feature in the Linux server. In this case, it is possible to use the RSA algorithm implemented inside of OpenSSL instead of the software fallback of libica. The appropriate configuration is shown in Example 33. This might have a smaller path length.

[6]RHEL 7.2: /etc/pki/tls/openssl.cnf, SLES 12 SP1: /etc/ssl/openssl.cnf


```
*** some lines not displayed ***
# DIGESTS
# - SHA1, SHA256 digests
#
#default_algorithms = ALL
default_algorithms = RAND,CIPHERS
```

Example 33: ibmca section in OpenSSL configuration file for an environment w/o access to CEX5S

If you wish to configure SSH clients and SSHD (as described in chapter 7.2.1 and 7.2.2) to ensure that only AES (and not TDES) is used as cipher suite, it might be an option to use the AES implementation inside OpenSSL instead of the implementation inside libica (i.e. omit CIPHERS keyword in the configuration for the ibmca engine).

For an environment with access toa CEX5S, we recommend that you have at least RSA and RAND enabled for the ibmca engine (see Example 34).

```
*** some lines not displayed ***
# DIGESTS
# - SHA1, SHA256 digests
#
#default_algorithms = ALL
default_algorithms = RAND,RSA
```

Example 34: ibmca section in OpenSSL configuration file for an environment with access to CEX5S

6 CPACF Support for OpenSSH

Disclaimer:
All numbers presented in the following section are not the result of official benchmark tests. These results might 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 run as guests in a shared z/VM environment.

In section 5.1 and 5.2, we described our environment and how to prepare it for using hardware crypto support including using support from Crypto Express feature. We also showed how we can check that RSA requests are executed in the crypto card. This was done to prove that hardware support of an available Crypto Express feature is used by our Linux servers. Using a Crypto Express feature is an optional possibility which might not be available for your Linux server. Therefore, we describe in the following, how you can test and verify, whether the acceleration support for encryption of CPACF is available in your Linux environment. The icastats command of libica shows whether the supported algorithms of libica are performed using hardware support or as software fallback. For this purpose, we use the default configuration of the ibmca engine with

```
default_algorithms = ALL
```

as shown in Example 20. In the following part, we describe how we can check that the hardware crypto support of the CPACF is used.

6.1 General test using openssl speed

For a first check of whether or not we can use the CPACF capabilities, we use the openssl speed command.

First, we reset the *icastats* counters, then we execute Tripel DES and AES encryption.

```
gnirss@zlx14020:~> icastats -r

gnirss@zlx14020:~> 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    149341.40k   379676.20k   625266.17k   748493.66k   786368.77k

gnirss@zlx14020:~> 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     166848.23k   404200.51k   966176.46k  1422325.97k  1536445.34k

gnirss@zlx14020:~> openssl speed -evp aes-192-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-192-cbc     167041.91k   459028.70k  1013489.78k  1370947.81k  1498835.35k

gnirss@zlx14020:~> openssl speed -evp aes-256-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-256-cbc     164065.00k   530333.48k  1062997.07k  1462723.25k  1571415.28k
```

Example 35: Perform TDES and AES encryption using openssl speed -evp <cipher> with libica

We check the counters and see that AES and Triple DES are using CPACF support (see Example 36).

```
gnirss@zlx14020:~> icastats
 function      |          # hardware          |        # software
               |       ENC     CRYPT    DEC  |       ENC     CRYPT    DEC
---------------+------------------------------+------------------------------
        SHA-1  |               176            |               0
      SHA-224  |                 0            |               0
      SHA-256  |                 0            |               0
      SHA-384  |                 0            |               0
      SHA-512  |                 0            |               0
        P_RNG  |                 4            |               0
       RSA-ME  |                 0            |               0
      RSA-CRT  |                 0            |               0
      DES ECB  |         0               0    |         0               0
      DES CBC  |         0               0    |         0               0
      DES OFB  |         0               0    |         0               0
      DES CFB  |         0               0    |         0               0
   DES CTRLST  |         0               0    |         0               0
     DES CMAC  |         0               0    |         0               0
     3DES ECB  |         0               0    |         0               0
     3DES CBC  |  55228978               0    |         0               0
     3DES OFB  |         0               0    |         0               0
     3DES CFB  |         0               0    |         0               0
 3DES CTRLIST  |         0               0    |         0               0
    3DES CMAC  |         0               0    |         0               0
      AES ECB  |         0               0    |         0               0
      AES CBC  | 205164189               0    |         0               0
      AES OFB  |         0               0    |         0               0
*** some lines not displayed ***
```

Example 36: Increased counters for TDES and AES encryption

This test demonstrates that in our environment, CPACF is working as expected. We summarize the throughput results of this test in Table 1 and we observe that we doubled the throughput compared to

Server	Cipher	With dyn. engine ibmca [MB/s]
z13	des-ede3-cbc	786368.77
z13	aes-128-cbc	1536445.34
z13	aes-192-cbc	1498835.35
z13	aes-256-cbc	1571415.28

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

a IBM z10^{TM} environment (see [1]) using a standard encryption tool.

6.2 First test with SCP of OpenSSH

Now we are interested in the question as to whether OpenSSH can use CPACF support in our test environment. As a first test, we use the SCP (Secure Copy) command to check for the usage of the underlying hardware crypto capabilities. We create a test file, which will be used to be copied with the SCP command. The file has to be large enough to enable us to clearly observe the occuring effects.

```
gnirss@zlx14020:~> dd if=/dev/zero of=testdata.txt bs=1048576 count=200
200+0 Datensätze ein
200+0 Datensätze aus
209715200 Bytes (210 MB) kopiert, 0,227014 s, 924 MB/s

gnirss@zlx14020:~> ls -lh testdata.txt
-rw-r--r-- 1 gnirss users 200M 13. Feb 18:39 testdata.txt
```

Example 37: Creation of a test file

Before we start the first test with SCP, we reset the counters to be able to determine after the test whether CPACF has been used.

```
gnirss@zlx14020:~> icastats -r

gnirss@zlx14020:~> icastats
 function      |          # hardware          |        # software
---------------+------------------------------+-------------------------------
               |       ENC     CRYPT    DEC  |         ENC     CRYPT    DEC
---------------+------------------------------+-------------------------------
         SHA-1 |                 0            |                 0
       SHA-224 |                 0            |                 0
       SHA-256 |                 0            |                 0
*** some lines not displayed ***
```

Example 38: Reset icastats counters and verify the result

Now we start to copy the data to the localhost, because for this test it is not necessary to send the test file via the network to any other server. We do not need to store the data after receiving them, therefore we specify /dev/null as receiving device for this test. At first we use TDES encryption,

```
gnirss@zlx14020:~> time scp -c 3des-cbc testdata.txt localhost:/dev/null
The authenticity of host 'localhost (::1)' can't be established.
ECDSA key fingerprint is 15:4c:da:ae:11:09:77:7b:94:c6:f5:34:c7:8f:b1:19 [MD5].
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'localhost' (ECDSA) to the list of known hosts.
Password:
testdata.txt                                  100%  200MB 200.0MB/s   00:01
```



```
real    0m14.688s
user    0m0.662s
sys     0m0.135s
```

Example 39: Secure Copy of test data with TDES encryption

then we test with AES cipher.

```
gnirss@zlx14020:~> time scp -c aes128-cbc testdata.txt localhost:/dev/null
Password:
testdata.txt                                  100%  200MB 200.0MB/s   00:00

real    0m4.999s
user    0m0.441s
sys     0m0.130s
```

Example 40: Secure Copy of test data with AES encryption

Again using the icastats command, we can see that the counter for TDES and AES counter has been increased. This shows that the CPACF support has been used for these ciphers (see Example 41).

```
gnirss@zlx14020:~> icastats
 function      |          # hardware          |        # software
---------------+------------------------------+-------------------------------
               |       ENC     CRYPT    DEC   |       ENC     CRYPT    DEC
---------------+------------------------------+-------------------------------
        SHA-1  |                80            |                0
      SHA-224  |                 0            |                0
      SHA-256  |                30            |                0
      SHA-384  |                 0            |                0
      SHA-512  |                 0            |                0
        P_RNG  |                 2            |                0
       RSA-ME  |                 0            |                0
      RSA-CRT  |                 0            |                0
      DES ECB  |         0               0    |        0               0
      DES CBC  |         0               0    |        0               0
      DES OFB  |         0               0    |        0               0
      DES CFB  |         0               0    |        0               0
   DES CTRLST  |         0               0    |        0               0
     DES CMAC  |         0               0    |        0               0
     3DES ECB  |         0               0    |        0               0
     3DES CBC  |     12805            2527    |        0               0
     3DES OFB  |         0               0    |        0               0
     3DES CFB  |         0               0    |        0               0
 3DES CTRLIST  |         0               0    |        0               0
    3DES CMAC  |         0               0    |        0               0
      AES ECB  |         0               0    |        0               0
      AES CBC  |     12754            2250    |        0               0
*** some lines not displayed ***
```

Example 41: Increased counters for TDES and AES after SCP of test data

We have now shown how easy it is to configure a Linux server on z Systems to use underlying acceleration support of CPACF for Secure Copy file transfer (SCP via OpenSSH).

6.3 Test with SSH client

Now we want to verify whether or not CPACF or CEX5S are also used during simple SSH session. To establish a SSH session to our Linux servers, we uses the SSH command from a Linux[7] workstation and specify the cipher and host key algorithm to be used. This allows us to check immediately whether hardware encryption support is used.

In the SSH session to our Linux server, the encryption of the traffic for the host part is done via the SSH daemon (SSHD)[8]. In our case, SSHD is running under the root userid and therefore we have to check the icastats counter of the root userid. We can use either icastats -A or icastats -U root. First, we reset the counters check and verify that all is zeroized (see Example 42) in an existing SSH session to our SLES server.

```
gnirss@zlx14020:~> sudo icastats -R
gnirss's password:

gnirss@zlx14020:~> sudo icastats -U root
 function      |          # hardware          |       # software
---------------+------------------------------+------------------------------
               |       ENC     CRYPT    DEC   |        ENC     CRYPT    DEC
---------------+------------------------------+------------------------------
        SHA-1  |                 0            |                 0
      SHA-224  |                 0            |                 0
      SHA-256  |                 0            |                 0
      SHA-384  |                 0            |                 0
      SHA-512  |                 0            |                 0
        P_RNG  |                 0            |                 0
       RSA-ME  |                 0            |                 0
      RSA-CRT  |                 0            |                 0
      DES ECB  |          0              0    |          0              0
      DES CBC  |          0              0    |          0              0
*** some lines not displayed ***
```

Example 42: Reset icatsts counters also for root and verify the result (on SLES)

Next, we open a second SSH session from the PC to our SLES server and we specify explicitly the Triple DES as cipher and RSA as host key algorithm to be used (see Example 43).

```
gnirss@manfred-W541:~$ ssh -c 3des -o HostKeyAlgorithms=ssh-rsa
gnirss@<our_Linux_SLES_server>
Password:
Last login: Tue Mar 29 11:42:46 2016 from <my_PC_address>
```

Example 43: Open second SSH session to SLES server using Triple DES and RSA

In the first session to our SLES server, using the icastats command we we can now see, that SHA, RSA as well as Triple DES counters for hardware have been increased (see Example 44). The activities with SHA and RSA are based on the opening of the SSH session.

```
gnirss@zlx14020:~> sudo icastats -U root
 function      |          # hardware          |       # software
---------------+------------------------------+------------------------------
               |       ENC     CRYPT    DEC   |        ENC     CRYPT    DEC
---------------+------------------------------+------------------------------
        SHA-1  |                438           |                 0
      SHA-224  |                 0            |                 0
```

[7] From a Windows PC, the putty command could be used. We do not discuss specific aspects of using putty in this paper. Especially for selecting specific ciphers, MACs and asymmetric algorithms, please refer to the documentation of putty.

[8] In our environment, the SSHD uses in any case the openssl config with dynamic engine support for ibmca enabled, as we know, that we have already rebooted the Linux server or restarted the SSHD service.


```
      SHA-256 |                14 |                 0
      SHA-384 |                 0 |                 0
      SHA-512 |                 0 |                 0
        P_RNG |                 1 |                 0
       RSA-ME |                 0 |                 0
      RSA-CRT |                 1 |                 0
      DES ECB |         0       0 |         0       0
      DES CBC |         0       0 |         0       0
      DES OFB |         0       0 |         0       0
      DES CFB |         0       0 |         0       0
   DES CTRLST |         0       0 |         0       0
     DES CMAC |         0       0 |         0       0
     3DES ECB |         0       0 |         0       0
     3DES CBC |        11      19 |         0       0
*** some lines not displayed ***
```

Example 44: Check for increased counters for the root user (on SLES)

Further activities in the second SSH session (like ls command, ...) will lead to an increase of the Triple DES counters.

In another test, we will now prove that in a SSH session, the AES algorithm also uses CPACF support. For this purpose, we repeat the above short test with AES cipher using our Red Hat RHEL 7.2 server. After reset and verification of the counters, we open a second SSH session to our RHEL server as described in Example 45,

```
gnirss@manfred-W541:~$ ssh -c aes256-cbc -o HostKeyAlgorithms=ssh-rsa
gnirss@<our_Linux_RHEL_server>
gnirss@<our_Linux_RHEL_server>'s password:
Last login: Tue Mar 29 07:28:03 2016 from <my_PC_address>
```

Example 45: Open second SSH session to RHEL server using AES and RSA

then we check for the icastats counters in the first session to our RHEL server and

```
[gnirss@zlb14020 ~]$ sudo icastats -U root
 function      |          # hardware          |          # software
---------------+------------------------------+------------------------------
               |       ENC     CRYPT     DEC  |       ENC     CRYPT     DEC
---------------+------------------------------+------------------------------
         SHA-1 |               411            |                 0
       SHA-224 |                 0            |                 0
       SHA-256 |                14            |                 0
       SHA-384 |                 0            |                 0
       SHA-512 |                 0            |                 0
         P_RNG |                 1            |                 0
        RSA-ME |                 0            |                 0
       RSA-CRT |                 0            |                 1
       DES ECB |         0       0            |         0       0
       DES CBC |         0       0            |         0       0
       DES OFB |         0       0            |         0       0
       DES CFB |         0       0            |         0       0
    DES CTRLST |         0       0            |         0       0
      DES CMAC |         0       0            |         0       0
      3DES ECB |         0       0            |         0       0
      3DES CBC |         0       0            |         0       0
      3DES OFB |         0       0            |         0       0
      3DES CFB |         0       0            |         0       0
  3DES CTRLIST |         0       0            |         0       0
     3DES CMAC |         0       0            |         0       0
```



```
      AES ECB |          0                0 |          0                0
      AES CBC |          9               17 |          0                0
*** some lines not displayed ***
```

Example 46: Check for increased counters for the root user (on RHEL)

we see that AES is performed with the support of CPACF, but that RSA is executed as software fallback (see Example 46). Even the device driver is loaded, and we know that RSA can be executed in the server using support from Crypto Express features (see Example 31). At the time of writing this paper, SSH session itself can not benefit from the underlying RSA hardware support in the current available version of RHEL. This issue is known and already addressed. A fix will be provided with a later corrective service or a later release of RHEL.

We have now shown, how easy it is to configure a Linux server on z Systems to use underlying acceleration support of CPACF for SSH session (via OpenSSL).

7 Selection of cipher and MAC

The SSH protocol allows various algorithms to be used for the authentication part (during the handshake), the encryption of the data (ciphers), and the integrity checking (Message Authentication Code).

Which cipher, Message Authentication Code (MAC) and asymmetric algorithms are used for an SSH connection that can be determined manually by the user, or by an automatic selection during establishment of the session partners (negotiation by the session partners depending on their configuration).

7.1 Small comparison between SHA with CPACF support and MD5

OpenSSH uses hash-based Message Authentication Codes (HMAC). CPACF provides support for SHA Message Authentication Code. In a pure software environment, MD5 is usually faster than SHA[9], and therefore MD5 is very often used as default.

Independent of the selection of a MAC for protection of user data integrity, there are some hashing operations during OpenSSH session negotiation. Mainly, there are SHA-1 and SHA-256 operations required for the key exchange. The MAC to be used for ensuring data integrity can be selected explicitly or via the search order in the SSH and SSHD configuration.

In the following three Examples (47, 48 and 49) we show the positive effect of using CPACF support for hashing.

```
gnirss@zlx14020:~> time scp -c aes128-cbc -o MACs=hmac-md5 testdata.txt
gnirss@localhost:/dev/null
Password:
testdata.txt                                  100%  200MB 200.0MB/s   00:01

real    0m4.471s
user    0m0.442s
sys     0m0.130s
```

Example 47: SCP: hashing with MD5 (software)

We can compare the effect of using MD5 (software only) versus SHA-1 (supported by CPACF)

```
gnirss@zlx14020:~> time scp -c aes128-cbc -o MACs=hmac-sha1 testdata.txt
gnirss@localhost:/dev/null
Password:
testdata.txt                                  100%  200MB 200.0MB/s   00:00
```

[9] SHA-1 can be considered weak in comparison with SHA-256 or SHA-512, but is still widely used for protecting data integrity. NIST, as well as other organizations (like BSI), recommend to stop using SHA-1 and migrate to algorithms of SHA-2 family.


```
real    0m3.539s
user    0m0.235s
sys     0m0.128s
```

Example 48: SCP: hashing with SHA-1 (using CPACF support)

and also versus SHA-256 (supported by CPACF)

```
gnirss@zlx14020:~> time scp -c aes128-cbc -o MACs=hmac-sha2-256 testdata.txt
gnirss@localhost:/dev/null
Password:
testdata.txt                                  100%  200MB 200.0MB/s   00:01

real    0m3.764s
user    0m0.214s
sys     0m0.136s
```

Example 49: SCP: hashing with SHA-2 (using CPACF support)

and we see that when we transfer test data using SHA, we need less CPU time (*user* and *sys*) compared to MD5. From a performance perspective, it has an advantage using SHA[10] with CPACF instead of MD5. For compatibility reasons, you might want to keep the MD5 algorithm in the list of available MACs. However, place the SHA algorithms in the first position of the search oder (see chapter 7.2.1).

7.2 Profiles for OpenSSH client and server

In most cases, it is not convenient to specify the desired ciphers and MACs with each SSH, SCP, SFTP, or rsync request. A better method is to adapt the profiles for SSH or SSHD to determine which algorithms are available and to determine the default search order. For performance reasons, it is recommended to place those algorithms at the top of the search order, which benefit from CPACF or CEX5S support (see also chapter 9.1 for more details). Note that in addition to performance aspects, enterprise policies and compliance regulations have to be considered and also have priority.

Not all ciphers and message authentication code (MAC) algorithms are supported by CPACF. To benefit from IBM z Systems CPACF support, an appropriate cipher and MAC should be selected when a SSH session is established. The SSH client and SSH server negotiate which cipher and which MAC will be used during the session. Both, client and server have a list of available ciphers and MACs. The client determines which cipher and MAC will be used depending on the available algorithms on the server and the client's preferences, according to the search order in the client's profile (see RFC4253 section 7.1). The list and the default search order can be adapted according to 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. From a performance perspective, we recommend that you place AES and TDES at the top of the search order for the symmetric ciphers.

The search order is important for all cases where a cipher or MAC is not explicitly specified when the user issues an *ssh* command. We assume that this will be the common case, and explicitly specifying a cipher or a MAC (as shown in Example 43 or 49) is an exception.

7.2.1 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 50) can be modified.

```
*** some lines not displayed ***
#   Protocol 2,1
#   Cipher 3des
```

[10] We do not see a big difference with our simple test between SHA-1 and SHA-256. Considering also performance aspects, algorithms of SHA-2 family are to be preferred


```
#    Ciphers aes128-ctr,aes192-ctr,aes256-ctr,arcfour256,arcfour128,
#       aes128-cbc,3des-cbc
#    MACs hmac-md5,hmac-sha1,umac-64@openssh.com,hmac-ripemd160
*** some lines not displayed ***
```

Example 50: Default search order in *ssh_config* file

According the man pages (*man ssh_config*), the default list of available symmetric ciphers and the default search oder is:

```
aes128-ctr,aes192-ctr,aes256-ctr,arcfour256,arcfour128,
aes128-gcm@openssh.com,aes256-gcm@openssh.com,
chacha20-poly1305@openssh.com,
aes128-cbc,3des-cbc,blowfish-cbc,cast128-cbc,aes192-cbc,
aes256-cbc,arcfour
```

The default list and search order for the MACs is:

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

In Example 51, we have modified the default by using the keywords *Ciphers* and *MACs* to change the search order and place algorithms at the top, which benefit from CPACF support.

```
*** some lines not displayed ***
#    Protocol 2,1
#    Cipher 3des
#    Ciphers aes128-ctr,aes192-ctr,aes256-ctr,arcfour256,arcfour128,
#       aes128-cbc,3des-cbc
Ciphers   aes256-ctr,aes192-ctr,aes128-ctr,aes256-gcm@openssh.com,
aes128-gcm@openssh.com,aes256-cbc,aes192-cbc,aes128-cbc,3des-cbc,
chacha20-poly1305@openssh.com,arcfour256,arcfour128,blowfish-cbc,
cast128-cbc,arcfour
#    MACs hmac-md5,hmac-sha1,umac-64@openssh.com,hmac-ripemd160
MACs hmac-sha2-256-etm@openssh.com,hmac-sha2-512-etm@openssh.com,
hmac-sha1-96-etm@openssh.com,hmac-sha1-etm@openssh.com,
hmac-sha2-256,hmac-sha2-512,hmac-sha1-96,hmac-sha1,
hmac-md5-etm@openssh.com,umac-64-etm@openssh.com,umac-128-etm@openssh.com,
hmac-ripemd160-etm@openssh.com,hmac-md5-96-etm@openssh.com,
hmac-md5,umac-64@openssh.com,umac-128@openssh.com,hmac-ripemd160,hmac-md5-96
*** some lines not displayed ***
```

Example 51: Modified search order in *ssh_config* file to benefit from CPACF

7.2.2 SSHD server configuration

To determine which algorithms can be used by the SSHD server, the configuration file */etc/ssh/sshd_config* of the server can be modified.

To specify the ciphers permitted, the keyword *Ciphers* (for protocol version 2) can be used. To specify the message authentication code algorithms permitted, which are used for data integrity protection, the keyword *MACs* (for protocol version 2) can be used in the configuration file. Multiple algorithms must be comma-separated. The order of the algorithms does not matter on the server side, as the client will select the first method in the client's search list that also appears on the server's list.

In our test environment, we modify the list of available algorithms in the *sshd_config* file using the *Ciphers* and *MACs* keywords, to allow only those algorithms which benefit from CPACF support of z Systems[11] (see Example 52). Note that a modification of the *sshd_config* file will only take effect after a restart of the SSHD daemon.

```
*** some lines not displayed ***
Ciphers  aes256-ctr,aes192-ctr,aes128-ctr,aes256-gcm@openssh.com,
aes128-gcm@openssh.com,aes256-cbc,aes192-cbc,aes128-cbc,3des-cbc
MACs hmac-sha2-256-etm@openssh.com,hmac-sha2-512-etm@openssh.com,
hmac-sha1-96-etm@openssh.com,hmac-sha1-etm@openssh.com,
hmac-sha2-256,hmac-sha2-512,hmac-sha1-96,hmac-sha1
*** some lines not displayed ***
```

Example 52: *sshd_config* file: modification to use CPACF support

8 Crypto Express support for RSA with OpenSSH

In chapter 6.3 we showed that OpenSSH can utilize RSA hardware cryptographic support from a Crypto Express feature. 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. Therefore, we do not spend much effort in studying the effect of using hardware support for RSA in terms of performance and throughput.

For a rough test, we create a very short file and use this file for Secure Copy.

```
gnirss@zlx14020:~> ls -lh testdata_short
-rw-r--r-- 1 gnirss users 2 29. Mär 18:38 testdata_short
```

Example 53: Small file to be used by SCP

After a reset of the icastats counters, we use the SCP command as indicated in Example 54 multiple times.

```
gnirss@zlx14020:~> time scp -c aes128-cbc -o HostKeyAlgorithms=ssh-rsa
testdata_short localhost:/dev/null
Password:
testdata_short                                100%    2     0.0KB/s   00:00

real    0m3.055s
user    0m0.011s
sys     0m0.002s
```

Example 54: Secure Copy of a small file using RSA

Then we verify using icastats that RSA is really executed in the hardware (see Example 55).

```
gnirss@zlx14020:~> icastats
 function      |          # hardware          |         # software
---------------+------------------------------+-----------------------------
               |       ENC     CRYPT    DEC   |       ENC     CRYPT    DEC
---------------+------------------------------+-----------------------------
         SHA-1 |               164            |                0
       SHA-224 |                 0            |                0
       SHA-256 |                56            |                0
       SHA-384 |                 0            |                0
```

[11] Allowing only algorithms which benefit from CPACF support might, or might not be applicable for general environments. There are regulations and other aspects to considered as well.


```
      SHA-512 |                0 |                0
        P_RNG |                4 |                0
       RSA-ME |                4 |                0
      RSA-CRT |                0 |                0
*** some lines not displayed ***
```

Example 55: Secure Copy of a small file using RSA

To compare the effect of execution RSA with hardware support and pure software execution in OpenSSL, we exclude RSA from the capabilities of the ibmca engine by adapting ibmca section in the configuration file of OpenSSL (see also chapter 5.3). Instead of

```
default_algorithms = ALL
```

we reduce the default to

```
default_algorithms = RAND, CIPHERS
```

as shown in Example 33. This is an easy method for a fast switch between Crypto Express support and software execution of RSA.

Next after resetting the icastats counters, we again repeatedly execute the SCP command as indicated in Example 54, and verify with the icastats counter that RSA is not executed with the support of the ibmca engine. As expected, after checking the execution time of user and sys, we cannot find any significant difference with our test case for a single[12] SCP request.

9 Some more performance aspects

Performance can be influenced by many factors that are related to cryptography dimensions. There are 4 cryptographic dimensions to take into account in order to secure services, and to guaranty data privacy.

A choice must be made for each of these dimensions:

- Cipher Algorithm
- Mode of Operation
- Key Size
- Key Protection profile (No covered in this article).

OpenSSH obey the same laws, and we have to correctly read OpenSSH Cipher: **aes256-cbc**

1. Algorithm is **AES**
2. Key size is **256**
3. Mode of Operation is **CBC** (Chain Block Cipher).

Let's see the performance impact in OpenSSH in relation to the 4 crypto dimensions.

9.1 Choice of cipher algorithm

Also called cipher, a cipher algorithm is a process to convert ordinary information (called plaintext) into unintelligible text (called ciphertext). An algorithm (a cipher) must be considered as secure and not deprecated or compromised. In the same algorithm family, some perform better than other even if these algorithms claim the same level of security.

Choice of cipher is important and choosing accelerated ciphers could lead to significant improvement factor. This effect is shown in Table 2 using SCP with a file with a size of 5 GB.

With OpenSSH, we can claim an improvement factor of more than 4 choosing a cipher that uses hardware-based acceleration (comparing cast128-cbc with aes128-gcm@openssh.com).

[12] Using RSA acceleration support of CEX5S will have a visible effect, when multiple requests are executed in parallel.
[13] non-standard non-RFC names have "@openssh.com"

Ciphers	Average Throughput [MB/s]	Times [Seconds]
cast128-cbc	96.5	51.8
blowfish-cbc	103.8	48.2
chacha20-poly1305@openssh.com[13]	117.9	42.4
arcfour	202.4	24.7
3des-cbc	233.1	21.4
aes128-ctr	275.3	18.2
aes192-ctr	279.6	17.9
aes256-ctr	280.1	17.9
aes256-cbc	282.0	17.7
aes128-cbc	315.1	15.9
aes192-cbc	315.2	15.9
aes256-gcm@openssh.com[13]	398.6	12.5
aes128-gcm@openssh.com[13]	417.9	12.0

Table 2: SCP of a 5 GB file: Algorithm matters

Ciphers	Average Throughput [MB/s]	Times [Seconds]
Case1		
aes128-ctr	275.3	18.2
aes192-ctr	279.6	17.9
aes256-ctr	280.1	17.9
Case2		
aes128-cbc	315.1	15.9
aes128-cbc	315.2	15.9
aes256-cbc	282.0	17.7
Case3		
aes128-qcm@openssh.com[13]	417.9	12.0
aes256-qcm@openssh.com[13]	398.6	12.5

Table 3: SCP of a 5 GB file: Key size matters

9.2 Choice of key size

The key size is usually a parameter of the algorithm, the longer is the key size, the stronger is the encryption. Key size impacts performance, as it improve the level of security. Key size matters, and can lead to significant degradation of performance.

Similar to Table 2, we have the focus in Table 3 on the key size.

- **Case1:** Changing key size for AES-CTR from 128 to 192 and 256.
- **Case2:** Changing key size for AES-CBC from 128 to 192 and 256.
- **Case3:** Changing key size for AES-GCM from 128 to 256.

With OpenSSH, choosing a bigger key could lead to performance degradation. In two of three cases, switching from AES with 128 bits key size to AES with 256 bits key size can result in a moderate performance degradation of about 10%. In case of doubt, please refer to your national key size recommendation or other regulations.

9.3 Choice of mode of operation

A mode of operation is an algorithm that uses a block cipher to provide an information service such as confidentiality or authenticity. A block cipher by itself is only suitable for the secure cryptographic transformation (encryption or decryption) of one fixed-length group of bits called a block. A mode of operation describes how to repeatedly apply a cipher's single-block operation to securely transform amounts of data larger than a block.

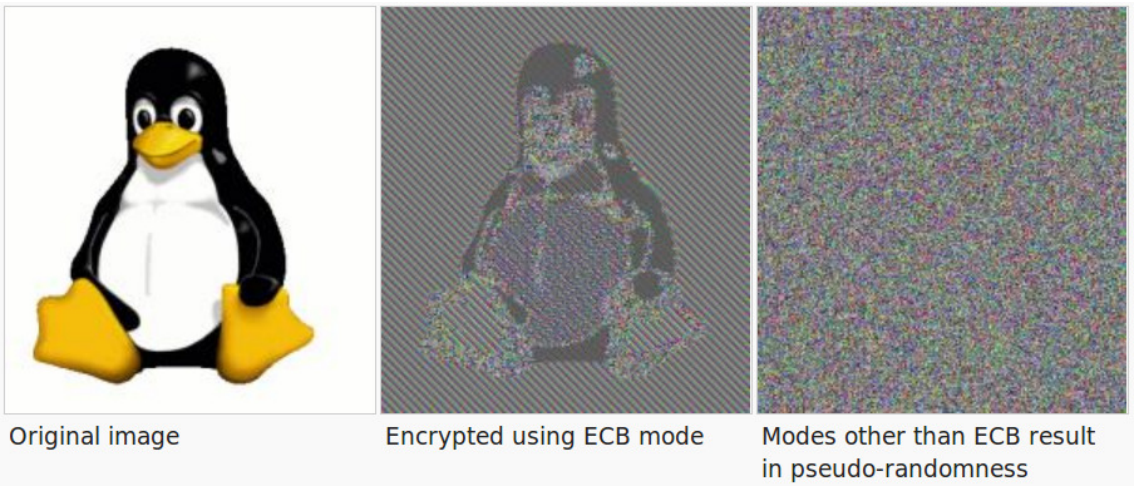

Figure 5: Influence of used Mode of Operation (source: Wikipedia, Block cipher mode of operation; Tux the Penguin, the Linux mascot created in 1996 by Larry Ewing (lewing@isc.tamu.edu) with The GIMP.)

According Figure 5, we see the limits of the simplest of the encryption modes - the Electronic Codebook (ECB) mode. The message is divided into blocks, and each block is encrypted separately. The weakness with ECB is that according the nature of the data to be encrypted, it is possible to identify some repetitive patterns (see also [10]).

There are alternative to ECB to provide encryption result in a pseudo-randomness. Each of these mode of operation has a different logic. Here is a list of common Mode of Operation:

- Cipher Block Chaining (CBC)
- Propagating Cipher Block Chaining (PCBC)
- Cipher Feedback (CFB)
- Output Feedback (OFB)
- Counter (CTR)

Similar to Table 2 and Table 3, in Table 4 the focus is on the mode of operation:

- **Case1:** Changing mode of operation for AES128 from ctr, to cbc and gcm.
- **Case2:** Changing mode of operation for AES192 from ctr, to cbc.
- **Case3:** Changing mode of operation for AES256 from ctr, to cbc and gcm.

With OpenSSH, as you can see CBC perform better than CTR. The performance delta can exceed 10%.

Ciphers	Average Throughput [MB/s]	Times [Seconds]
Case1		
aes128-**ctr**	275.3	18.2
aes128-**cbc**	315.1	15.9
aes128-qcm@openssh.com[13]	417.9	12.0
Case2		
aes192-**ctr**	279.6	17.9
aes192-**cbc**	315.2	15.9
Case3		
aes256-**ctr**	280.1	17.9
aes256-**cbc**	282.0	17.7
aes256-qcm@openssh.com[13]	398.6	12.5

Table 4: SCP of a 5 GB file: Mode of Operation matters

9.4 Choice of crypto key protection profile (optional)

This section describes the mechanism that is chosen to secure the application key. Basically,the application key appears in cleartext in the system memory (Clear Key). Using z Systems it is also possible to use Protected Key and Secure Key protection profile in order to ensure that even briefly the application key never appears in cleartext in the system memory. Protected Key uses a wrapped key to encrypt the application key, and this wrapped key is stored in HSA, not accessible from the operating system. Secure Key uses a Master Key to encrypt application key, and this master key is stored in an HSM (for z13: CEX5C or CEX5P), a temper-proof area.

In this paper, we do not discuss this aspect in any detail.

10 Conclusion

Using modern Linux distributions on z Systems, relatively low effort is required in order to enable a Linux Server to use existing hardware capabilities to accelerate encryption operations for OpenSSH. This advantage applies not only to SSH sessions, but also to Secure Copy (SCP), SFTP and rsync. Using hardware support for encryption, in particular the CPACF capabilities speeds up encryption operations and saves a lot of cycles. To benefit from this advantage, either specify explicitly ciphers and MACs with CPACF support, or adapt SSH and SSHD profiles and place those algorithms supported by CPACF at the top of the search order.

The team who wrote this paper

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

Uwe Denneler is a Senior IT Specialist in the IBM Client Center in the IBM Boeblingen Lab, Germany. He has more than 20 years of experience in the mainframe and IBM z/OS® field. He works with independent software vendor (ISV) and customer projects on z Systems (IBM z/OS, IBM z/VM, IBM z/VSE®, Linux on z Systems, and various subsystems). He also prepares demonstrations on IBM z Systems.

Harald Freudenberger is a Senior IT Specialist at the IBM Lab in Boeblingen, Germany. He works since more than 20 years in different areas around Linux development from the embedded world up to Linux for z Systems mainframes. 3 years ago he joined the Linux for z Systems crypto team and contributes to the platform crypto stack from Linux kernel up to higher crypto libraries like opencryptoki.

Paul Gallagher is a freelance Technical Writer who has worked in the IBM Lab in Boeblingen, Germany for over 20 years. He is currently working in different areas of Linux development (mainly KVM for IBM z Systems), but previously worked for many years on z/VSE development.

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

Guillaume Hoareau is a Certified IT Architect at the IBM Client Center, Montpellier, France. Holding a Master Degree in Computer Science, Guillaume joined IBM in 2005 as security expert, accompanying his customers in their digital transformation and ambitious security projects. Currently he is involved in several z Systems crypto Proof-of-Concept and Proof-of-Technology projects to promote fast adoption of leading edge technology such as blockchain, elliptic curve cryptography and data tokenization.

Arwed Tschoeke is a member of zATS team located at the IBM Client Center in the Boeblingen Lab. As Client Technical Architect his focus areas are LinuxONE, Cloud, virtualization solutions across multiple platforms and Linux. He is located in Hamburg, Germany.

Ingo Tuchscherer is a Senior IT Specialist at the IBM Lab in Boeblingen, Germany. He has more than 10 years of experience in the IBM mainframe environment. He started at IBM as a Software engineer for smart card operation systems, before he changed to the mainframe area to work as a Performance analyst. He continued his work as a software developer for Virtualization & Systems Management. Currently he works as a Software engineer for Cryptography & Security on Linux for z Systems and he is responsible for the crypto device driver and library.

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 currently is a member of the test team for Linux on z Systems. He is one of the leaders in the Boeblingen test community for years, teaching test fundamentals and he is now involved in the test of cryptographic functionality and solutions for Linux on IBM z Systems.

Acknowledgement

Our very best acknowledgement for discussions and helpful hints belongs to
Reinhard Bündgen, Frank Heimes, Brian W. Hugenbruch, Elisabeth Puritscher, Patrick Steuer, Richard Young,
as well as to Andy Polyakov and to Agnès Gnirss.

Acronyms

3DES Triple DES

AES Advanced Encryption Standard

AP Adjunct Processor

CBC Cipher Block Chaining

CEX Crypto Express feature

CEX5A Crypto Express 5 feature configured in accelerator mode

CEX5C Crypto Express 5 feature configured in coprocessor mode

CEX5P Crypto Express 5 feature configured in EP11 mode

CEX5S Crypto Express 5 feature

CFB Cipher Feedback

CPACF Central Processor Assist for Cryptographic Functions

CPU Central Processing Unit

CTR Counter

DES Data Encryption Standard

EP11 IBM Enterprise PKCS#11

ECB Electronic Codebook

FTP File Transfer Protocol

HMC Hardware Management Console

HSA Hardware System Area

HSM Hardware Security Module

IPL Initial Program Load

LPAR Logical partition

LIC Licensed Internal Code

MAC Message Authentication Code

MD5 Message-Digest algorithm 5

OFB Output Feedback

PCBC propagating Cipher Block Chaining

PRNG Pseudo Random Number Generator

RHEL Red Hat Enterprise Linux

RSA Rivest, Shamir and Adleman algorithm

SCP Secure Copy Protocol

SE Support Element

SELinux Security-Enhanced Linux

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

References

[1] First experiences with hardware Cryptographic Support for OpenSSH with Linux for z Systems, by Manfred Gnirss, Winfried Münch, Klaus Werner, and Arthur Winterling.
`http://ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP101690`

[2] IBM z13, Features & Benefits
`http://www.ibm.com/systems/z/hardware/z13_features.html`

[3] z Systems, Processor Resource/Systems Manager Planning Guide, SB10-7162-01

[4] Linux on z Systems, libica Programmer's Reference, Version 2.6, SC34-2602-07

[5] IBM z13 Configuration Setup SG24-8260-00, 2015

[6] IBM z13 Technical Guide SG24-8251-01, 2016

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

[8] z/VM V6.3 CP Planning and Administration, SC24-6178-09

[9] z Systems Hardware Management Console Operations Guide, Version 2.13.1
`https://www-304.ibm.com/servers/resourcelink/lib03010.nsf/pagesByDocid/0351070EB1B67CD985257F7000487D13?OpenDocument`
`http://www.ibm.com/support/knowledgecenter/HW11P_2.13.1/z13_kc_ditamaps/z13_v2r13m1_welcome.html`

[10] Wikipedia, Block cipher mode of operation
`https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation`

Trademarks

