Hardware cryptographic support for IBM Z and IBM LinuxONE with Ubuntu Server

Klaus Bergmann, Reinhard Buendgen, Uwe Denneler, Jonathan Furminger, Frank Heimes, Manfred Gnirss, Christian Rund, Patrick Steuer, Arwed Tschoeke

August 10, 2017

Abstract

This article summarizes our experiences with the setup, configuration and usage of OpenSSL, PKCS#11 and its related components for exploiting hardware-assisted cryptographic operations on IBM LinuxONE and IBM Z for clear key operations. The required steps are described, as well as findings in the areas of performance improvement using OpenSSH, Apache HTTP server and IBM Java. Based on our positive experiences we recommend that you should make use of these capabilities whenever performing cryptographic workloads on Ubuntu Server for IBM Z and IBM LinuxONE.

Contents

1 Introduction **1**

2 Hardware cryptographic support of IBM Z **1**
2.1 Verification of installed LIC 3863 using the SE . . . 2
2.2 Verification of installed LIC 3863 using a Linux command . . . 3
2.3 Configuration of Crypto Express feature for IBM Z . . . 5

3 Cryptographic support in Linux for z Systems (IBM Z) **5**
3.1 OpenSSL for LinuxONE and Linux for z Systems (IBM Z) . . . 6
3.2 PKCS#11 for LinuxONE and Linux for z Systems (IBM Z) . . . 7

4 Our hardware and software environment **7**

5 Installation of Ubuntu Server 16.04 LTS for OpenSSL **9**
5.1 Configuring ibmca engine . . . 16
5.2 Hardware cryptographic support for OpenSSL . . . 17
5.3 General test using openssl speed . . . 17
5.4 First test with SCP of OpenSSH . . . 19
5.5 Test with SSH client . . . 22
5.6 Exploiting hardware crypto support of AES and SHA without using the ibmca engine . . . 25
5.7 Selection of cipher and MAC for OpenSSH . . . 33
5.7.1 Using SHA with CPACF support versus MD5 . . . 33
5.7.2 Profiles for OpenSSH client and server . . . 33
5.7.3 SSHD server configuration . . . 34
5.7.4 SSH client configuration . . . 35
5.8 Crypto Express support for RSA with OpenSSH . . . 36
5.9 Apache on Ubuntu - using mod_ssl . . . 37
5.9.1 Prerequisite tasks . . . 37
5.9.2 Configuring OpenSSL . . . 37
5.9.3 Configuring Apache . . . 38
5.9.4 Choosing SSL/TLS cipher suites . . . 39
5.9.5 Starting the web server . . . 40

6 Configuring PKCS#11 environment **41**
6.1 Installation and preparation of openCryptoki . . . 41
6.1.1 Configuration of the openCryptoki ICA token . . . 44
6.1.2 Configuration of the openCryptoki software token . . . 46
6.2 Verify the configuration of openCryptoki . . . 46
6.3 Apache on Ubuntu - using mod_nss . . . 51
6.4 Using IBM Java with hardware cryptographic support on Ubuntu . . . 51
6.4.1 Installation of IBM Java on Ubuntu . . . 52
6.4.2 Enable IBM Java for using strong encryption . . . 52
6.4.3 Hardware support for encryption in a IBM Java 7 environment . . . 53
6.4.4 Hardware support for encryption in a IBM Java 8 environment . . . 58
6.4.5 IBM Java 8: Using hardware acceleration for AES and RSA with two providers . . . 61

7 Conclusion **62**

Source code of java program sample **63**

The team who wrote this paper **64**

 ©Copyright IBM Corporation 2017

Acknowledgement **64**

Acronyms **64**

References **67**

Trademarks **68**

List of Figures

1 IBM z13: LIC 3863 is installed 3
2 Hardware support for cryptographic stack of LinuxONE and Linux for z Systems (IBM Z) 6
3 PKCS#11 architecture 41
4 JCA architecture 51
5 Selection of algorithms out of multiple providers 52

List of Tables

1 Throughput for 8 KB blocks encrypted with *openssl speed -evp <cipher>* on IBM z13 or IBM LinuxONE Emperor 18

 ©Copyright IBM Corporation 2017

1 Introduction

It's no secret that 'security' became the most important topic that nowadays concerns all C-level executives (see [1]). And it's not only about the security of the most valuable assets many companies have: the data, but also about avoiding bad publicity due to data breaches - like we unfortunately heard too often in the (IT-) news these days. The risk is high - not only for smaller companies, but also for big enterprises, ISPs and even global web companies!

Security in information technology is a broad field and covers:

- Authentication - to ensure identity (certificates)
- Key Exchange - to exchange cryptographic keys and do handshaking
- Confidentiality - to ensure a message can only be read by a desired receiver (encryption)
- Integrity - to ensure that a received message is still the original one and wasn't altered (hash/MAC)
- Nonrepudiation - to ensure that a message really came from a certain sender (signature)

These functions are largely handled by the clear key cryptography discipline. Clear key is the most common mode of performing cryptography and indicates that the key is handled in clear at some level inside an operating system and software stack.

However, pervasive encryption doesn't come for free. It requires a solid planning, proper implementation and even then still ongoing effort, in reviews, audits and operation. From a performance point of view encryption is expensive and can heavily impact performance, throughput, CPU load and the overall system utilization. But all servers of the family IBM Z® provide hardware encryption support that can be used to mitigate the impact of expensive encryption operations.

Since version 4.4 (release in September 2006) OpenSSH supports dynamic engine load of OpenSSL, which enables OpenSSH to benefit from IBM Z cryptographic hardware support [1].

This document describes how to setup hardware accelerated encryption with OpenSSL and our experiences based on Ubuntu® Server 16.04 LTS™ running on IBM z Systems z13™ and IBM LinuxONE Emperor™ hardware[2], as well as findings about performance and throughput, concerning Apache® HTTP server and IBM Java™.

The following test scenarios and examples are based on the IBM z Systems z13 platform and an Ubuntu Server 16.04 LTS Linux distribution.

This article extends and supersedes the prior article [2] and covers the throughput improvements over the past years, the topic PKCS#11 and focuses exclusively on Ubuntu Server 16.04. However certain tasks like for example the basic OpenSSH setup is similar to the description in article [3].

Note, aspects of AppArmor or SELinux are not covered in this paper.

2 Hardware cryptographic support of IBM Z

Servers of the IBM Z family provide 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. 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

[1] OpenSSH package needs to be compiled with flag *−−with−ssl−engine* to use this support, see https://www.openssh.com/txt/release-4.4. This is reflected in all modern linux for z Systems distributions.

[2] Identical setup, functionality and behaviour occur also when using IBM z Systems z13s™ or IBM LinuxONE Rockhopper™. Only performance differences might occur.

 ©Copyright IBM Corporation 2017

are for clear key operations (this means, the cryptographic key is provided by application software in plain text 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) for CCA operations, or in EP11 mode (CEX5P) as Coprocessor for PKCS#11 compatible secure key cryptography. 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 the Licensed Internal Code (LIC) feature 3863 (Crypto Enablement feature), which is available free of charge (see also [4], [5]). By default, IBM Z is delivered to customers without this feature, unless it is ordered explicitly by the customer. The installation of this feature a is non-disruptive operation.

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)[3].

Open the Hardware Management Console (HMC) web user interface in your browser and

- select *Tasks Index*
- find or filter for *Single Object Operations*
- switch to the Support Element (SE) by selecting *Single Object Operations*
- select your z System, and confirm with *OK*
- confirm establishing a session with *Yes*
- select again *Tasks Index* at the SE
- find or filter for *System Details*
- select *System Details*
- select your system, and confirm with *OK*
- and check for the phrase *CP Assist for Crypto functions: Installed* or *CP Assist for Crypto functions: Not installed* (see Figure 1).

[3]Here we do not describe and discuss the new way for configuration of LPARs with Dynamic Partition Management (DPM).

 ©Copyright IBM Corporation 2017

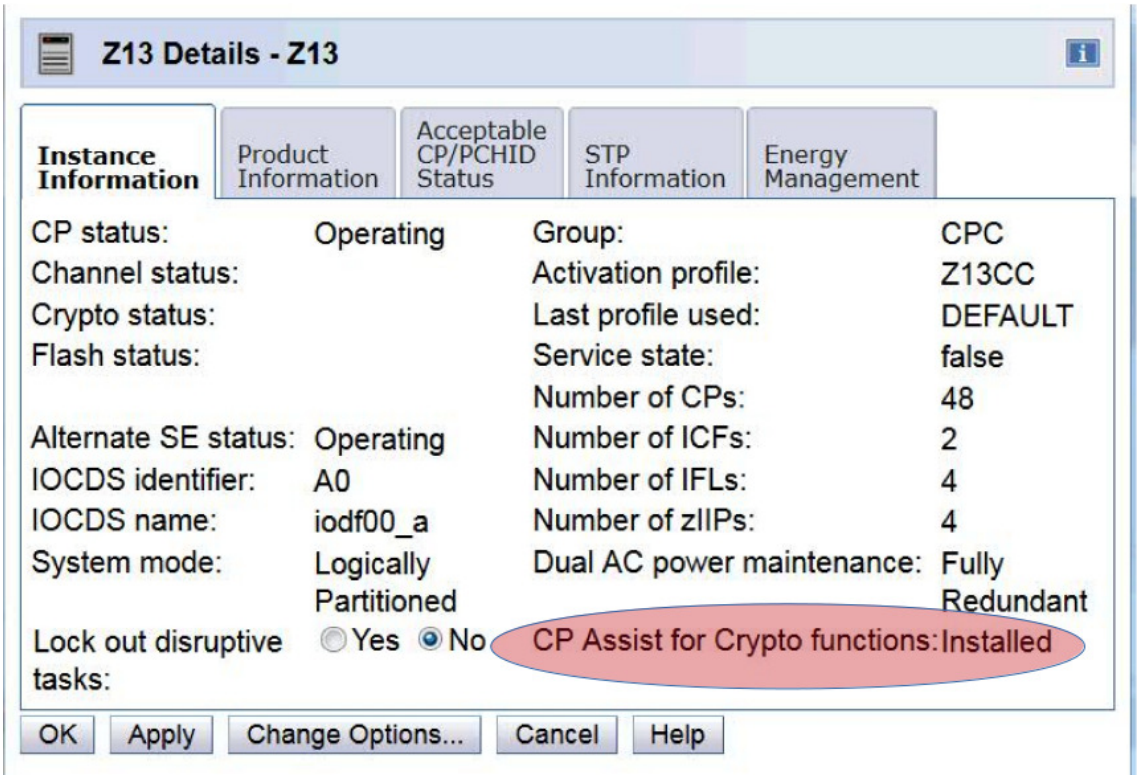

Figure 1: IBM z13: LIC 3863 is installed

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-utils* package is installed on the Linux for z Systems server, it will automatically install the dependent *libica2* package as well.

```
sudo apt --yes install libica-utils
```

Example 1: Installation of libica-utils

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

```
ubuntu@zlin42:~$ 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
```

 ©Copyright IBM Corporation 2017

```
       RSA CRT |      no      |      yes
       DES ECB |      no      |      yes
. . .
```

Example 2: 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[4].

```
ubuntu@zlin42:~$ 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 XTS |     yes    |      no
```

Example 3: Encryption algorithms supported in CPACF of IBM z13

If you find a *no* in column *# software* in the output of the *icainfo* command (see Example 3), there is no software fallback implemented in libica (see also chapter 6 in [6]).

The output of the *icainfo* command can be limited to the relevant DES and AES functions like this:

[4]The *no* for RSA ME and RSA CRT support in the column *# hardware* of Example 3 indicates that there is no access from the Linux server to a Crypto Express feature, or that the crypto device driver is not loaded.

 ©Copyright IBM Corporation 2017

```
ubuntu@zlin42:~$ $ icainfo | head -n 4 && icainfo | grep '\(AES\|DES\)'
The following CP Assist for Cryptographic Function (CPACF)
operations are supported by libica on this system:
 function       | # hardware | # software
----------------+------------+------------
       DES ECB  |     yes    |     yes
       DES CBC  |     yes    |     yes
       DES OFB  |     yes    |      no
       DES CFB  |     yes    |      no
       DES CTR  |     yes    |      no
      DES CMAC  |     yes    |      no
      3DES ECB  |     yes    |     yes
      3DES CBC  |     yes    |     yes
      3DES OFB  |     yes    |      no
      3DES CFB  |     yes    |      no
      3DES CTR  |     yes    |      no
     3DES CMAC  |     yes    |      no
       AES ECB  |     yes    |     yes
       AES CBC  |     yes    |     yes
       AES OFB  |     yes    |      no
       AES CFB  |     yes    |      no
       AES CTR  |     yes    |      no
      AES CMAC  |     yes    |      no
       AES XTS  |     yes    |      no
```

Example 4: Filtered output of icainfo

2.3 Configuration of Crypto Express feature for IBM Z

If you have a Crypto Express5 (CEX5S) adapter in your IBM Z or LinuxONE™ machine, 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 [7] and chapter 6 of [8]. For details how to enable access to the CEX feature for a Linux system running in a z/VM® environment, see chapter 6 of [9] and [10]. In [11], information about how to work with the HMC can be found.

A brief overview of the LPAR crypto configuration steps:

- Open the HMC web user interface in your browser
- Select *Systems Management*, an IBM Z machine via its id and the LPAR you want to modify
- Now select *Operational Customization* and *Change LPAR Cryptographic Controls*
- At *Assigned Cryptos* choose *Select Action* and then *Add*
- Finally specify the *Assigned Cryptos* - specify at least one AP as *Candidate and Online*

3 Cryptographic support in Linux for z Systems (IBM Z)

In a Linux environment, there are basically two standard interfaces for cryptographic support, which can used by middleware and applications:

- OpenSSL
- PKCS#11

 ©Copyright IBM Corporation 2017

Both interfaces with their appropriate libraries and services are included in LinuxONE and Linux for z Systems (IBM Z) distributions.

In Figure 2 we see an overview of the LinuxONE and Linux for z Systems crypto stack. This overview contains components for clear key, protected key and secure key cryptographic support. The scope of this paper is limited to clear key cryptography, therefore we only look at the objects in the figure marked in green. From the application layer point of view, cryptographic requests are typically processed by using standard crypto interfaces: An application use directly or indirectly OpenSSL or PKCS#11 libraries to perform the cryptographic work. We do not miss to mention, that there are some services in these cryptographic interfaces (ICC, JCA/JCE) which bypass OpenSSL and PKCS#11 libraries for some specific operations and invoke directly hardware supported crypto services of the CPACF.

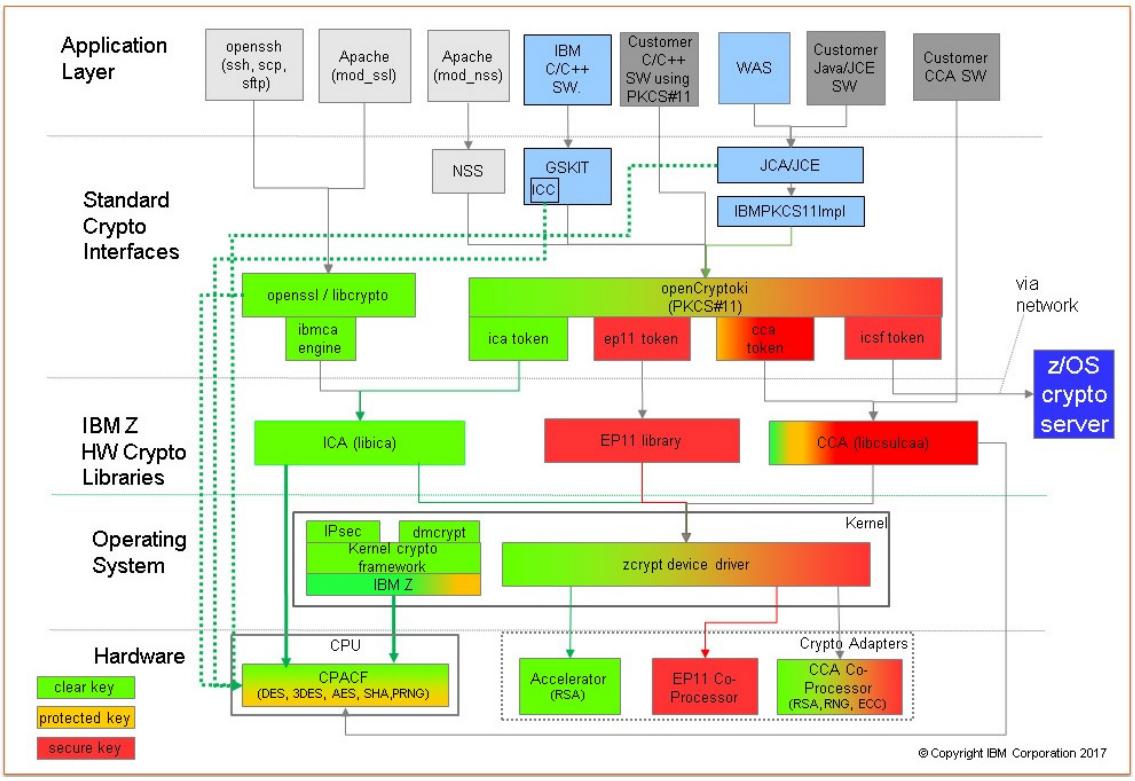

Figure 2: Hardware support for cryptographic stack of LinuxONE and Linux for z Systems (IBM Z)

3.1 OpenSSL for LinuxONE and Linux for z Systems (IBM Z)

In an IBM Z environment, you can install the ibmca engine and configure OpenSSL for dynamic engine loading[5]. In this case, OpenSSL does not perform all encryption requests by itself, but passes those supported by the engine 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 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[6]. 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

[5]This paper only covers using *ibmca* engine for OpenSSL. Aspects of using other engines are not discussed.

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

to a Crypto Express feature. You need to adapt the guest entry in the z/VM directory, if you intend to access the Crypto Express feature from Linux (see chapter 6 of [9]).

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, if 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 fast due to the use of CPACF. But he even implemented the software fallback for AES and SHA in assembler code for the case LIC 3863 is not installed (see chapter 2).

3.2 PKCS#11 for LinuxONE and Linux for z Systems (IBM Z)

The PKCS#11 interface is another standard, which allows applications to use cryptographic services in a standardized manner. Applications can use encryption services executed in software or also access services which are based on cryptographic devices. The PKCS#11 standard unifies the way in which applications access cryptographic objects. To achieve this, so called *tokens* and *slots* are used.

IBM provides an implementation of the PKCS#11 Interface with the openCryptoki, which is Open Source and shipped with the LinuxONE and Linux for z Systems distributions. Besides software cryptographic services, openCryptoki enables applications to exploit hardware support of the Z architecture for encryption, if the ICA token is configured for openCryptoki. Cryptographic requests to openCryptoki can be passed via the ICA token to the CPACF for symmetric encryption and pseudo random number generations or for RSA support to the CEX5S (if available).

4 Our hardware and software environment

For our test, we use Linux servers as guests[7] in a z/VM LPAR of a IBM z13, as well as directly installed in a LPAR.

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 (installed by default)
- openssl (installed by default)
- openssl-ibmca (installation required to achieve IBM Z hardware crypto exploitation)
- libica2 (installation required to achieve IBM Z hardware crypto exploitation)
- zcrypt driver (device driver, provided as kernel module)

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 and up and running with a default installation.

```
ubuntu@zlin42:~$ lscpu
Architecture:          s390x
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Big Endian
CPU(s):                4
On-line CPU(s) list:   0-3
Thread(s) per core:    1
Core(s) per socket:    1
```

[7] 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.

 ©Copyright IBM Corporation 2017

```
Socket(s) per book:    1
Book(s):               4
NUMA node(s):          1
Vendor ID:             IBM/S390
BogoMIPS:              3033.00
Hypervisor:            z/VM 6.3.0
Hypervisor vendor:     IBM
Virtualization type:   full
Dispatching mode:      horizontal
L1d cache:             128K
L1i cache:             96K
L2d cache:             2048K
L2i cache:             2048K
NUMA node0 CPU(s):     0-63
Flags:                 esan3 zarch stfle msa ldisp eimm dfp etf3eh highgprs
```

Example 5: Our environment - system running as z/VM guest

```
ubuntu@zlin43:~$ lscpu
Architecture:          s390x
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Big Endian
CPU(s):                4
On-line CPU(s) list:   0-3
Thread(s) per core:    2
Core(s) per socket:    8
Socket(s) per book:    3
Book(s):               8
NUMA node(s):          1
Vendor ID:             IBM/S390
BogoMIPS:              3033.00
Hypervisor:            PR/SM
Hypervisor vendor:     IBM
Virtualization type:   full
Dispatching mode:      horizontal
L1d cache:             128K
L1i cache:             96K
L2d cache:             2048K
L2i cache:             2048K
NUMA node0 CPU(s):     0-255
Flags:                 esan3 zarch stfle msa ldisp eimm dfp edat etf3eh highgprs
                       te vx
```

Example 6: Our environment - system running in LPAR

We use two systems - one running as a z/VM guest and the other running directly in a LPAR, but both running the same Ubuntu Server 16.04 LTS installed with the latest updates. The z/VM directory for our Linux guest contains the CRYPTO statement to assign a dedicated crypto queue of a CEX5C. We use (see Example 7) domain 5 of adapter 0.

```
USER ZLIN42 <password> 2G 4G G
. . .
* crypto
   CRYPT DOMAIN 5 APDED 0
. . .
```

Example 7: Extract of z/VM directory entry for Linux guests with dedicated access to CEX5S

 ©Copyright IBM Corporation 2017

Note that when using Crypto Express with OpenSSH/OpenSSL, we could alternatively use a virtualized cryptographic adapter for acceleration of the RSA handshake. For this case CRYPTO APVIRTUAL in the guest definition in the user directory is sufficient for RSA (clear key) acceleration.

5 Installation of Ubuntu Server 16.04 LTS for OpenSSL

We use a default installation of Ubuntu Server 16.04 LTS with the latest updates. There is no need to specify anything special for IBM Z hardware cryptographic exploitation during install time.

After the default installation is finished, it's recommended to update the repository index and to install any potential updates with:

```
ubuntu@zlin42:~$ sudo apt update
ubuntu@zlin42:~$ sudo apt upgrade
```

Example 8: Our environment (Ubuntu Server 16.04 LTS) - system update and upgrade

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

```
ubuntu@zlin42:~$ uname -a
Linux zlin42 4.4.0-49-generic #70-Ubuntu SMP Fri Nov 11 16:44:28 UTC 2016 s390x
s390x s390x GNU/Linux
```

Example 9: Our environment (Ubuntu Server 16.04 LTS) - system and kernel

Notice that the maintenance level may differ, because it changes over time while the support and maintenance of Ubuntu Server 16.04 LTS is going on.

```
ubuntu@zlin42:~$ lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 16.04.1 LTS
Release:        16.04
Codename:       xenial
```

Example 10: Our environment (Ubuntu Server 16.04 LTS) - LSB information

```
ubuntu@zlin42:~$ cat /etc/os-release
NAME="Ubuntu"
VERSION="16.04.1 LTS (Xenial Xerus)"
ID=ubuntu
ID_LIKE=debian
PRETTY_NAME="Ubuntu 16.04.1 LTS"
VERSION_ID="16.04"
HOME_URL="http://www.ubuntu.com/"
SUPPORT_URL="http://help.ubuntu.com/"
BUG_REPORT_URL="http://bugs.launchpad.net/ubuntu/"
VERSION_CODENAME=xenial
UBUNTU_CODENAME=xenial
```

Example 11: Our environment (Ubuntu Server 16.04 LTS) - version/release of operating system

Depending on the time and date of the installation or update, your system may be described as 16.04 LTS, 16.04.1 LTS, ..., 16.04.5 LTS. These so called *point releases* mark the different refresh levels that will be released during the 5 year support of an Ubuntu Server LTS release (see [12]).

The following packages that are required for encryption, including hardware cryptographic support:

- openssl
- openssh-server

 ©Copyright IBM Corporation 2017

- openssh-client
- openssl-ibmca
- libica-utils
- libica2

The following three packages need to be installed for IBM Z hardware cryptographic support:

```
ubuntu@zlin42:~$ sudo apt-get install openssl-ibmca libica-utils libica2
```

Example 12: Our environment (Ubuntu Server 16.04 LTS) - additional packages to install

It's good practice to verify that all needed packages are properly installed (see Example 13):

```
ubuntu@zlin42:~$ dpkg -l openssl openssh-server openssh-client openssl-ibmca
   libica-utils libica2
Desired=Unknown/Install/Remove/Purge/Hold
| Status=Not/Inst/Conf-files/Unpacked/halF-conf/Half-inst/trig-aWait/Trig-pend
|/ Err?=(none)/Reinst-required (Status,Err: uppercase=bad)
||/ Name              Version                 Arch. Description
+++-==================-=======================-=====-==================================
ii  libica-utils      2.6.1-1ubuntu2          s390x  hardware cryptography support
                                                    for Linux on z Systems (utils)
ii  libica2:s390x     2.6.1-1ubuntu2          s390x  hardware cryptography support
                                                    for IBM System z hardware
ii  openssh-client    1:7.2p2-4ubuntu2.1      s390x  secure shell (SSH) client, for
                                                    secure access to remote
                                                    machines
ii  openssh-server    1:7.2p2-4ubuntu2.1      s390x  secure shell (SSH) server, for
                                                    secure access from remote
                                                    machines
ii  openssl           1.0.2g-1ubuntu4.5       s390x  Secure Sockets Layer toolkit
                                                    - cryptographic utility
ii  openssl-ibmca     1.3.0-0ubuntu2.16.04.1  s390x  libica based hardware
                                                    acceleration engine for OpenSSL
```

Example 13: Our environment (Ubuntu Server 16.04 LTS) - additional packages to install

At this point it's still only the default engine of OpenSSL available:

```
ubuntu@zlin42:~$ openssl engine
(dynamic) Dynamic engine loading support
ubuntu@zlin42:~$ openssl engine -c
(dynamic) Dynamic engine loading support
```

Example 14: Our environment (Ubuntu Server 16.04 LTS) - Engine ibmca is not yet available for OpenSSL

To make use of the ibmca engine and to benefit from the implemented hardware support, the configuration file of OpenSSL need to be modified. To customize the OpenSSL configuration to enable dynamic engine loading for ibmca, perform the following steps:

1. Take a backup of the configuration file before you change it.

```
ubuntu@zlin42:~$ ls -la /etc/ssl/openssl.cnf
-rw-r--r-- 1 root root 10835 Sep 23 14:22 /etc/ssl/openssl.cnf
ubuntu@zlin42:~$ sudo cp -p /etc/ssl/openssl.cnf
/etc/ssl/openssl.cnf_backup$(date +%Y%m%d)
ubuntu@zlin42:~$ ls -la /etc/ssl/openssl.cnf*
-rw-r--r-- 1 root root 10835 Sep 23 14:22 /etc/ssl/openssl.cnf
-rw-r--r-- 1 root root 10835 Sep 23 14:22 /etc/ssl/openssl.cnf_backup20170127
```

Example 15: Take a backup of original configuration

 ©Copyright IBM Corporation 2017

2. Append the ibmca related configuration lines to the OpenSSL configuration file.

```
ubuntu@zlin42:~$ ls -la /etc/ssl/openssl.cnf
-rw-r--r-- 1 root root 10835 Sep 23 14:22 /etc/ssl/openssl.cnf
ubuntu@zlin42:~$ ls -la /usr/share/doc/openssl-ibmca/examples/
   openssl.cnf.sample
-rw-r--r-- 1 root root 1416 Nov 18 00:15 /usr/share/doc/openssl-ibmca/
examples/openssl.cnf.sample
ubuntu@zlin42:~$ sudo tee -a /etc/ssl/openssl.cnf
   < /usr/share/doc/openssl-ibmca/examples/openssl.cnf.sample
. . .
ubuntu@zlin42:~$ ls -la /etc/ssl/openssl.cnf
-rw-r--r-- 1 root root 12251 Jan 27 10:57 /etc/ssl/openssl.cnf
```

Example 16: Append ibmca section to the configuration

Notice, this changed file size, date and time of openssl.cnf file.

3. Verify that there is an ibmca_section at the end of the OpenSSL configuration file.

```
ubuntu@zlin42:~$ grep -n ibmca_section /etc/ssl/openssl.cnf
368:ibmca = ibmca_section
371:[ibmca_section]
```

Example 17: ibmca section exists at end of the configuration

Notice that the reference to the ibmca section and the section itself exist.

4. Insert the following line

```
openssl_conf = openssl_def
```

Example 18: Necessary line at the top of the configuration file to enable ibmca engine

at the top of the configuration file and ensure that this line appears only once in the configuration file. Hence check for any lines that contain this setting and comment these out. Afterwards insert a line number 10 at the beginning with this setting. For this purpose, you may just execute the following two lines:

```
ubuntu@zlin42:~$ sudo sed -i 's/^\(openssl_conf = openssl_def.*$\)/# \1/g'
   /etc/ssl/openssl.cnf
ubuntu@zlin42:~$ sudo sed -i '10i openssl_conf = openssl_def'
   /etc/ssl/openssl.cnf
```

Example 19: Insert line with *openssl_conf = openssl_def* to enable *ibmca* engine

Finally verify that there is only one line left (line number 10) with that pattern:

```
ubuntu@zlin42:~$ grep -n "^openssl_conf = openssl_def" /etc/ssl/openssl.cnf
10:openssl_conf = openssl_def
```

Example 20: Verify for line with *openssl_conf = openssl_def*

Notice that the configuration file should now look like in Example 22.

5. You may verify the value of the dynamic_path variable and in case needed adjust accordingly.

```
ubuntu@zlin42:~$ grep dynamic_path /etc/ssl/openssl.cnf
# Set the dynamic_path to where the libibmca.so engine
dynamic_path = /usr/lib/s390x-linux-gnu/openssl-1.0.0/engines/libibmca.so
ubuntu@zlin42:~$ ls -la /usr/lib/s390x-linux-gnu/openssl-1.0.0/
   engines/libibmca.so
```

 ©Copyright IBM Corporation 2017

```
-rw-r--r-- 1 root root 47936 Nov 17 18:15 /usr/lib/s390x-linux-gnu/
   openssl-1.0.0/engines/libibmca.so
```

Example 21: Verify dynamic path variable

Notice that the reference to the library and the library itself is existing.

```
ubuntu@zlin42:~$ sudo vi /etc/ssl/openssl.cnf
#
# 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
openssl_conf = openssl_def          #     <== line inserted

# Extra OBJECT IDENTIFIER info:
#oid_file               = $ENV::HOME/.oid
oid_section             = new_oids

. . .

# 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          #     <== line commented

[openssl_def]
engines = engine_section

[engine_section]

ibmca = ibmca_section

[ibmca_section]

# The openssl engine path for libibmca.so.
# Set the dynamic_path to where the libibmca.so engine
# resides on the system.
dynamic_path = /usr/lib/s390x-linux-gnu/openssl-1.0.0/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.
```

 ©Copyright IBM Corporation 2017

```
#
# 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, SHA512 digests
#
default_algorithms = ALL
#default_algorithms = RAND,RSA,DSA,DH,CIPHERS,DIGESTS
```

Example 22: OpenSSL configuration file with dynamic engine loading support for ibmca

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

```
ubuntu@zlin42:~$ openssl engine
(dynamic) Dynamic engine loading support
(ibmca) Ibmca hardware engine support
```

Example 23: *ibmca* is part of the OpenSSL's dynamic engine list

We can also see the supported algorithms:

```
ubuntu@zlin42:~$ openssl engine -c
(dynamic) Dynamic engine loading support
(ibmca) Ibmca hardware engine support
 [RAND, DES-ECB, DES-CBC, DES-OFB, DES-CFB, DES-EDE3, DES-EDE3-CBC,
DES-EDE3-OFB, DES-EDE3-CFB, AES-128-ECB, AES-192-ECB, AES-256-ECB,
AES-128-CBC, AES-192-CBC, AES-256-CBC, AES-128-OFB, AES-192-OFB,
AES-256-OFB, AES-128-CFB, AES-192-CFB, AES-256-CFB, SHA1, SHA256, SHA512]
```

Example 24: Dynamic engine support for ibmca is is enabled for ciphers available via CPACF support

In case the system runs under z/VM the availability of the crypto queue can be verified with the following command:

```
ubuntu@zlin42:~$ sudo vmcp q v crypto
AP 000 CEX5C Domain 005 dedicated
```

Example 25: Access to a crypto queue is available (Domain 5)

Notice that access to Crypto Express hardware is available, in the way it has been defined in the z/VM directory (see Example 7). The crypto device driver needed for accessing the Crypto Express adapter is not yet loaded:

```
ubuntu@zlin42:~$ sudo lszcrypt
ubuntu's password:
lszcrypt: error - cryptographic device driver zcrypt is not loaded!
```

Example 26: Crypto device driver not loaded

Therefore all RSA requests will be executed as software fallback in libica:

 ©Copyright IBM Corporation 2017

```
ubuntu@zlin42:~$ icainfo | grep -A 3 CPACF && icainfo | grep RSA
The following CP Assist for Cryptographic Function (CPACF)
operations are supported by libica on this system:
 function      | # hardware | # software
---------------+------------+------------
       RSA ME |      no    |     yes
      RSA CRT |      no    |     yes
```

Example 27: RSA crypto not yet hardware enabled

Note that in order to use the *vmcp* and *lszcrypt* command, the package *s390-tools* has to be installed, which is mandatory for Ubuntu on s390x anyway.. For our z/VM guest, the z/VM privilege class G has been assigned (see also Example 7) and the guest is allowed to submit some commands to the underlying hipervisor.

To load the crypto device driver, use the modprobe command:

```
ubuntu@zlin42:~$ sudo modprobe ap
```

Example 28: Load the crypto device driver

and verify whether it was successful (see Example 29):

```
ubuntu@zlin42:~$ lsmod | grep ap
zcrypt_api             32768  2 zcrypt_cex4,zcrypt_msgtype6
ap                     36864  3 zcrypt_cex4,zcrypt_api,zcrypt_msgtype6
```

Example 29: Verify load of crypto device driver

Ensure that the device driver will be from now on automatically loaded by default after a re-IPL (re-boot):

```
ubuntu@zlin42:~$ echo 'ap' | sudo tee -a /etc/modules
ap

ubuntu@zlin42:~$ grep ap /etc/modules
ap

ubuntu@zlin42:~$ sudo update-initramfs -k all -u
```

Example 30: Load crypto device driver automatically

Now the *lszcrypt* command shows that access to the crypto device is available (see Example 31).

```
ubuntu@zlin42:~$ sudo lszcrypt -V
card00: CEX5C       online
```

Example 31: Crypto device driver is loaded and accessible

Since the crypto device driver is now loaded, also indicated by the *icainfo* command, the hardware support for RSA ME and RSA CRT is now available via libica library (see Example 32 and compare with Example 3 and 27).

```
ubuntu@zlin42:~$ icainfo | grep -A 3 CPACF && icainfo | grep RSA
The following CP Assist for Cryptographic Function (CPACF)
operations are supported by libica on this system:
 function      | # hardware | #software
---------------+------------+------------
       RSA ME |     yes    |     yes
      RSA CRT |     yes    |     yes
```

Example 32: RSA is available via hardware support

Now we check again the support of the dynamic engine (see Example 33)

 ©Copyright IBM Corporation 2017

```
ubuntu@zlin42:~$ 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-192-ECB, AES-256-ECB,
AES-128-CBC, AES-192-CBC, AES-256-CBC, AES-128-OFB, AES-192-OFB, AES-256-OFB,
AES-128-CFB, AES-192-CFB, AES-256-CFB, SHA1, SHA256, SHA512]
```

Example 33: Dynamic engine support for ibmca is is enabled for ciphers available via CPACF and CEX5S support

and we see, that *ibmca* engine now supports additionally RSA, DSA and DH after the crypto device driver has been loaded (compare with Example 24).

This is consistent with the information available in sysfs:

```
ubuntu@zlin42:~$ ls /sys/devices/ap
card00  module  power  uevent
ubuntu@zlin42:~$ 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 34: sysfs with support for cryptographic adapter

We see that the cryptographic adapter is online ("1" in Example 35)

```
ubuntu@zlin42:~$ cat /sys/devices/ap/card00/online
1
```

Example 35: Cryptographic adapter is online

And that the Crypto Express cryptographic adapter is a CEX5S ("11" in Example 36), that is configured in coprocessor mode ("CEX5C" in example 36).

```
ubuntu@zlin42:~$ cat /sys/devices/ap/card00/hwtype
11
ubuntu@zlin42:~$ cat /sys/devices/ap/card00/type
CEX5C
```

Example 36: Crypto Express5 cryptographic adapter configured in coprocessor mode

The number of executed requests in the cryptographic adapter can now be checked (see Example 37). A change of this counter will be observed if RSA requests using the cryptographic adapter are executed.

```
ubuntu@zlin42:~$ cat /sys/devices/ap/card00/request_count
1
```

Example 37: Number of requests that are already processed by this device

Let's perform crypto operations that use the cryptographic adapter, especially RSA:

```
ubuntu@zlin42:~$ openssl speed rsa2048 -elapsed
You have chosen to measure elapsed time instead of user CPU time.
Doing 2048 bit private rsa's for 10s: 10544 2048 bit private RSA's in 10.00s
Doing 2048 bit public rsa's for 10s: 40387 2048 bit public RSA's in 10.00s
OpenSSL 1.0.2g  1 Mar 2016
built on: reproducible build, date unspecified
. . .
                  sign    verify    sign/s verify/s
rsa 2048 bits 0.000948s 0.000248s   1054.4   4038.7
```

Example 38: Test for RSA requests

 ©Copyright IBM Corporation 2017

After the test is completed an increased number of requests can now be identified (see Example 39). This means, the Crypto Express feature has been used.

```
ubuntu@zlin42:~$ cat /sys/devices/ap/card00/request_count
50935
```

Example 39: Number of requests that are processed by this device

Alternatively, the *icastats* command can be used to verify whether RSA uses hardware crypto support via libica or not (see Example 40):

```
ubuntu@zlin42:~$ icastats | head -n 4 && icastats | grep RSA
 function      |          # hardware        |        # software
---------------+---------------------------+-------------------------
               |       ENC     CRYPT    DEC |        ENC     CRYPT    DEC
---------------+---------------------------+-------------------------
       RSA-ME  |           40388           |                 0
      RSA-CRT  |           10545           |                 0
```

Example 40: RSA requests performed with hardware support in Ubuntu Server

Instead of checking the information in sysfs, you can use the option *-VV* or *-VVV* of the *lszcrypt* command:

```
ubuntu@zlin42:~$ lszcrypt -VVV
card00: CEX5C        online   hwtype=11 depth=7 request_count=50935
   pendingq_count=0 requestq_count=0 functions=0x92000000
```

Example 41: Crypto device information with increased verbose level

5.1 Configuring ibmca engine

In the *ibmca* section of the OpenSSL configuration file[8], it is possible to 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 RAND, RSA, DSA, DH, MACs, or the symmetric ciphers.

We mentioned already in chapter 3 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 22) to exclude all DIGESTS (see Example 42).

```
user@zlin42:~$ tail -n 5 /etc/ssl/openssl.cnf
# DIGESTS
# - SHA1, SHA256, SHA512 digests
#
#default_algorithms = ALL
default_algorithms = RAND,RSA,DSA,DH,CIPHERS
```

Example 42: ibmca section 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 43. This might have a shorter path length.

```
user@zlin42:~$ tail -n 5 /etc/ssl/openssl.cnf
# DIGESTS
# - SHA1, SHA256, SHA512 digests
#
```

[8] Ubuntu Server 16.04: /etc/ssl/openssl.cnf

 ©Copyright IBM Corporation 2017

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

Example 43: 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 section 5.7.3 and 5.7.4) to ensure that only AES (and not 3DES) 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 to a CEX5S configured as CEX5A or CEX5C, we recommend that you have at least RSA, DSA, DH and RAND enabled for the ibmca engine (see Example 44).

```
user@zlin42:~$ tail -n 5 /etc/ssl/openssl.cnf
# DIGESTS
# - SHA1, SHA256, SHA512 digests
#
#default_algorithms = ALL
default_algorithms = RAND,RSA,DSA,DH
```

Example 44: ibmca section in OpenSSL configuration file for an environment with access to CEX5S

5.2 Hardware cryptographic support for OpenSSL

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 chapter 5 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 cryptographic adapter. 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 and therefore a Crypto Express cryptographic adapter might not be available in 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:

```
user@zlin42:~$ grep "default_algorithms = ALL" /etc/ssl/openssl.cnf
default_algorithms = ALL
```

Example 45: The ibmca engine is allowed to use all its supported algorithms

as shown in Example 22. In the following part, we describe how we can check that the hardware crypto support of the CPACF is used.

5.3 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 Triple DES (3DES) and AES encryption.

```
user@zlin42:~$ icastats -r

user@zlin42:~$ openssl speed -evp des-ede3-cbc 2>/dev/null | tail -n 3
The 'numbers' are in 1000s of bytes per second processed.
type             16 bytes     64 bytes    256 bytes   1024 bytes   8192 bytes
des-ede3-cbc    128248.33k   345060.07k   597404.50k   736006.83k   785044.82k
```

 ©Copyright IBM Corporation 2017

Server	Cipher	With dyn. engine ibmca [MB/s]
z13	des-ede3-cbc	785.0
z13	aes-128-cbc	1496.8
z13	aes-192-cbc	1574.3
z13	aes-256-cbc	1665.9

Table 1: Throughput for 8 KB blocks encrypted with *openssl speed -evp <cipher>* on IBM z13 or IBM LinuxONE Emperor

```
user@zlin42:~$ openssl speed -evp aes-128-cbc 2>/dev/null | tail -n 3
The 'numbers' are in 1000s of bytes per second processed.
type             16 bytes     64 bytes    256 bytes   1024 bytes   8192 bytes
aes-128-cbc     138935.81k   359826.77k   922169.94k  1317677.06k  1496801.28k

user@zlin42:~$ openssl speed -evp aes-192-cbc 2>/dev/null | tail -n 3
The 'numbers' are in 1000s of bytes per second processed.
type             16 bytes     64 bytes    256 bytes   1024 bytes   8192 bytes
aes-192-cbc     138315.88k   401957.03k   889071.79k  1292803.75k  1574322.18k

user@zlin42:~$ openssl speed -evp aes-256-cbc 2>/dev/null | tail -n 3
The 'numbers' are in 1000s of bytes per second processed.
type             16 bytes     64 bytes    256 bytes   1024 bytes   8192 bytes
aes-256-cbc     136824.28k   454427.43k   859805.27k  1298550.78k  1665933.31k
```

Example 46: Perform 3DES and AES encryption using openssl speed -evp <cipher> with libica

We check the counters and see that AES and 3DES are using CPACF support (see Example 47).

```
user@zlin42:~$ icastats
 function      |          # hardware        |        # software
-------------+-------------------------+-------------------------
              |       ENC    CRYPT    DEC |        ENC    CRYPT    DEC
-------------+-------------------------+-------------------------
        SHA-1 |              472          |                0
      SHA-224 |                0          |                0
. . .
        P_RNG |                0          |                0
 DRBG-SHA-512 |              676          |                0
       RSA-ME |                0          |                0
. . .
     3DES ECB |          0              0 |          0              0
     3DES CBC |   49602134              0 |          0              0
     3DES OFB |          0              0 |          0              0
. . .
      AES ECB |          0              0 |          0              0
      AES CBC |  178866082              0 |          0              0
      AES OFB |          0              0 |          0              0
. . .
```

Example 47: Increased counters for TDES and AES encryption

This test demonstrates that in our environment, CPACF is working as expected - means z Systems hardware cryptography is indeed used. We summarize the throughput results of this test in Table 1 and we observe that we doubled the throughput compared to a IBM z10^{TM} environment (see [2]) using a standard encryption tool.

 ©Copyright IBM Corporation 2017

5.4 First test with SCP of OpenSSH

The next question to answer is whether OpenSSH can use CPACF support in our test environment or not. As a first test, we use the SCP (Secure Copy) command to check for the usage of the underlying hardware crypto capabilities.

Note, in the *ibmca* section of the OpenSSL configuration file we have

```
default_algorithms = ALL
```

specified (see also Examples 22 and 45) to used the full capabilities of the *ibmca* engine.

For not disturbing the counters of icastats with activities in the SSH session, we logon to the host using a cipher which does not make any benefit from CPACF support after the session is established:

```
user@workstation:~$ ssh -c chacha20-poly1305@openssh.com ubuntu@zlin42
ubuntu@zlin42's password:
Welcome to Ubuntu 16.04.1 LTS (GNU/Linux 4.4.0-51-generic s390x)
```

At first we create a test file, which will be used to be copied with the SCP command. The file has to be large enough to allow a clear observation of the occurring effects.

```
ubuntu@zlin42:~$ dd if=/dev/zero of=testdata.txt bs=1048576 count=200
200+0 records in
200+0 records out
209715200 bytes (210 MB, 200 MiB) copied, 0.0982474 s, 2.1 GB/s

ubuntu@zlin42:~$ ls -lh testdata.txt
-rw-rw-r-- 1 ubuntu ubuntu 200M Feb  5 12:11 testdata.txt
```

Before we start the first test with SCP, we reset the counters to be able to determine after the test whether CPACF or CEX5S has been used or not.

```
ubuntu@zlin42:~$ icastats -r
```

and we verify that all the counters shown by icastats command are 0 (see Example 48).

```
ubuntu@zlin42:~$ icastats
 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
        GHASH |                0           |                0
        P_RNG |                0           |                0
 DRBG-SHA-512 |                0           |                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 CTR |          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
```

 ©Copyright IBM Corporation 2017

```
     3DES CTR |          0              0 |          0              0
    3DES CMAC |          0              0 |          0              0
      AES ECB |          0              0 |          0              0
      AES CBC |          0              0 |          0              0
      AES OFB |          0              0 |          0              0
      AES CFB |          0              0 |          0              0
      AES CTR |          0              0 |          0              0
     AES CMAC |          0              0 |          0              0
      AES XTS |          0              0 |          0              0
```

Example 48: All counters of icastats are zero

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,

```
ubuntu@zlin42:~$ time scp -c 3des-cbc testdata.txt localhost:/dev/null
Unable to negotiate with ::1 port 22: no matching cipher found. Their offer:
   chacha20-poly1305@openssh.com,aes128-ctr,aes192-ctr,aes256-ctr,
   aes128-gcm@openssh.com,aes256-gcm@openssh.com
lost connection

real    0m0.028s
user    0m0.000s
sys     0m0.001s
```

Example 49: Secure Copy of test data with TDES encryption is here per default not supported

In our environment, using a workstation with an Ubuntu as SSH client and a host server (Ubuntu) as server, both without any modifications of defaults for SSH or SSHD, TDES as cipher for SCP is not supported. If TDES should be used for any reason, then the profiles for SSH and SSHD should be adapted (see section 5.7.2). We verify the *icastats* counters and see

```
ubuntu@zlin42:~$ icastats
 function      |          # hardware        |        # software
-------------+-------------------------+-------------------------
              |       ENC     CRYPT    DEC |        ENC     CRYPT    DEC
-------------+-------------------------+-------------------------
        SHA-1 |              120           |                0
      SHA-224 |                0           |                0
. . .
        P_RNG |                0           |                0
 DRBG-SHA-512 |              169           |                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 CTR |          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 CTR |          0              0 |          0              0
    3DES CMAC |          0              0 |          0              0
. . .
```

Example 50: Failed SCP with TDES increases some counters - but not the counters for TDES

 ©Copyright IBM Corporation 2017

that even we could not perform successfully the SCP command the counters, for SHA-1 and DRBG-SHA-512 have been increased. This is as of the activities during the handshake.

Now we reset the counters and try AES with a supported Counter Mode (CTR)

```
ubuntu@zlin42:~$ icastats -r
ubuntu@zlin42:~$ time scp -c aes256-ctr testdata.txt localhost:/dev/null
ubuntu@zlin42's password:
testdata.txt                                    100%  200MB 200.0MB/s   00:01

real    0m6.726s
user    0m0.365s
sys     0m0.169s
```

Example 51: Secure Copy of test data with AES256-CTR encryption

and check the counters.

```
ubuntu@zlin42:~$ icastats
 function      |          # hardware        |        # software
-------------+-------------------------+-------------------------
              |       ENC    CRYPT    DEC |        ENC    CRYPT    DEC
-------------+-------------------------+-------------------------
        SHA-1 |              124          |                0
      SHA-224 |                0          |                0
      SHA-256 |               17          |                0
      SHA-384 |                0          |                0
. . .
        P_RNG |                0          |                0
 DRBG-SHA-512 |              169          |                0
       RSA-ME |                0          |                0
      RSA-CRT |                0          |                0
. . .
      AES ECB |          0              0 |          0              0
      AES CBC |          0              0 |          0              0
      AES OFB |          0              0 |          0              0
      AES CFB |          0              0 |          0              0
      AES CTR |          0              0 |          0              0
     AES CMAC |          0              0 |          0              0
      AES XTS |          0              0 |          0              0
```

Example 52: SCP with AES CTR increases some counters but not AES CTR counter

We can see (Example 52), the counters of SHA are slightly higher than in the previous case (Example 50), where a SCP command could not be finished successfully, as of missing support. But we do not see any increase in the counter for AES CTR.

Even there is support for the AES CTR included in the library *libica* to be performed with the help of CPACF, for Secure Copy using the engine *ibmca* the AES CTR requests are not performed via *libica* support, as AES CTR is not supported via the *ibmca* engine (check for the supported ciphers of *ibmca* engine in Examples 24 and 33). Therefore the AES CTR operations are performed inside of OpenSSL.

As already mentioned, the OpenSSL code can use CPACF support for SHA and AES (see chapter 3). To proof, that also in the case for SCP using the cipher *aes256-ctr* the CPACF support is used, we can use the command cpacfstats as shown in section 5.6.

A small cross check confirms above statement. Also

```
ubuntu@zlin42:~$ openssl speed -evp aes-256-ctr 2>/dev/null | tail -n 3
```

does not increase AES CTR counter in the output of icastats. The library *libica* supports AES CTR, but the dynamic engine *ibmca* does not support CTR mode, therefore it is handled in OpenSSL code directly.

 ©Copyright IBM Corporation 2017

5.5 Test with SSH client

Now we want to verify whether or not CPACF or CEX5S are also used during a SSH session. To establish a SSH session to our Linux servers, we uses the SSH command from a Linux[9] 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)[10]. In our case, SSHD is running under the root userid and therefore we have to check the *icastats* counter of the root userid. For this purpose we can use either *icastats -A* or *icastats -U root*.

Note, in the ibmca section of the OpenSSL configuration file we have

```
default_algorithms = ALL
```

specified (see also Examples 22 and 45) to used the full capabilities of the ibmca engine.

Establish a SSH session between a client on a workstation and user on Linux host, the encryption and decryption of the SSH traffic on the host is done by the SSHD, which runs under the root user. To observe encryption operations performed via *libica* library, we can use command *icastats*. Note, we have to ensure that issuing *icastats* command inside the SSH session itself does not increases the counters (the observation should not disturb the behaviour, which we want to observe). For this purpose we open a session with a cipher which does not use SHA or AES

```
user@workstation:~$ ssh -c chacha20-poly1305@openssh.com ubuntu@zlin42
```

and we reset all *icastats* counters for all users, including for *root* using the *-R* option of the *icastats* command

```
ubuntu@zlin42:~$ sudo icastats -R
```

and then using the *-A* option we verify that all counters for all users are reset (in our test here, there are no other users, so only *ubuntu* and *root* will appear):

```
ubuntu@zlin42:~$ sudo icastats -A
user: ubuntu
 function      |          # hardware       |        # software
---------------+---------------------------+-------------------------
               |       ENC     CRYPT   DEC |        ENC     CRYPT   DEC
---------------+---------------------------+-------------------------
        SHA-1  |                0          |                0
. . .
      AES XTS  |         0               0 |         0               0
user: root
 function      |          # hardware       |        # software
---------------+---------------------------+-------------------------
               |       ENC     CRYPT   DEC |        ENC     CRYPT   DEC
---------------+---------------------------+-------------------------
        SHA-1  |                0          |                0
. . .
      AES XTS  |         0               0 |         0               0
```

We verify, that activities in this session (like *ls* command or also *sudo icastats -A*) do not affect the counters. The *icastats* counters remain unchanged.

Now we use agin the cipher *-c chacha20-poly1305@openssh.com* to open a second SSH session from the workstation to the host.

[9] From a Windows® workstation, 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.

[10] In our environment, the SSHD uses in any case the OpenSSL configuration with dynamic engine support for ibmca enabled, as we have already rebooted the Linux server or restarted the SSHD service.

 ©Copyright IBM Corporation 2017

```
user@workstation:~$ ssh -c chacha20-poly1305@openssh.com ubuntu@zlin42
```

After this second SSH session is established, we check in the first session for changed counters:

```
ubuntu@zlin42:~$ sudo icastats -A
user: ubuntu
 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
        GHASH |                0           |                0
        P_RNG |                0           |                0
 DRBG-SHA-512 |                0           |                0
       RSA-ME |                0           |                0
      RSA-CRT |                0           |                0
      DES ECB |         0               0 |         0               0
. . .
user: root
 function      |          # hardware        |        # software
-------------+-------------------------+-------------------------
              |       ENC    CRYPT    DEC |        ENC    CRYPT    DEC
-------------+-------------------------+-------------------------
        SHA-1 |              425           |                0
      SHA-224 |                0           |                0
      SHA-256 |               40           |                0
      SHA-384 |                0           |                0
      SHA-512 |                0           |                0
        GHASH |                0           |                0
        P_RNG |                0           |                0
 DRBG-SHA-512 |              169           |                0
       RSA-ME |                0           |                0
      RSA-CRT |                1           |                0
      DES ECB |         0               0 |         0               0
. . .
```

We can observe increased counters in root user for SHA-1, DRBG-SHA-512 and RSA CRT. Obviously RSA algorithm has been used for the handshake while establishing a session.

Activities inside the second session (like *ls* commands, etc.) do not increase any further the counters, neither of the user root, nor of the used userid ubuntu. This behaviour is as expected, as we chose a cipher for the symmetric encryption which does not make use of hardware acceleration of CPACF.

After logout of the second SSH session, we reset all the counters again in the first session:

```
ubuntu@zlin42:~$ sudo icastats -R
```

Now we check, which ciphers are supported in our SSH client on the Ubuntu workstation:

```
user@workstation:~$ ssh -Q cipher
3des-cbc
blowfish-cbc
cast128-cbc
arcfour
arcfour128
arcfour256
aes128-cbc
```

 ©Copyright IBM Corporation 2017

```
aes192-cbc
aes256-cbc
rijndael-cbc@lysator.liu.se
aes128-ctr
aes192-ctr
aes256-ctr
aes128-gcm@openssh.com
aes256-gcm@openssh.com
chacha20-poly1305@openssh.com
```

Example 53: Check for supported ciphers for the SSH client on Ubuntu workstation

We try to use explicitly a cipher (*-c 3des-cbc*) which is supported by CPACF

```
user@workstation:~$ ssh -c 3des-cbc ubuntu@zlin42
no matching cipher found: client 3des-cbc server chacha20-poly1305@openssh.com,
   aes128-ctr,aes192-ctr,aes256-ctr,aes128-gcm@openssh.com,aes256-gcm@openssh.com
```

In our environment, the host server (Ubuntu) does not support TDES as a cipher for SSH sessions without any modifications of defaults for SSHD. If TDES should be used for any reason, then the profile for SSHD should be adapted (see section 5.7.2). We verify the *icastats* counters (in the first session) and see

```
ubuntu@zlin42:~$ sudo icastats -A
. . .
user: root
 function      |          # hardware       |        # software
-------------+-------------------------+-------------------------
              |       ENC    CRYPT   DEC |        ENC    CRYPT   DEC
-------------+-------------------------+-------------------------
        SHA-1 |              281          |                0
      SHA-224 |                0          |                0
      SHA-256 |                7          |                0
      SHA-384 |                0          |                0
      SHA-512 |                0          |                0
        GHASH |                0          |                0
        P_RNG |                0          |                0
 DRBG-SHA-512 |              169          |                0
       RSA-ME |                0          |                0
      RSA-CRT |                0          |                0
. . .
```

some increased counters for SHA-1 and DRBG-SHA-512 as of the try to establish a session. Now we reset again the counters

```
ubuntu@zlin42:~$ sudo icastats -R
```

and for opening a second SSH session we use a supported cipher (*-c aes256-ctr*) and specify explicitly RSA algorithm (*-o HostKeyAlgorithms=ssh-rsa*) for the handshake as shown in Example 54.

```
user@workstation:~$ ssh -c aes256-ctr
   -o HostKeyAlgorithms=ssh-rsa ubuntu@zlin42
```

Example 54: Open second SSH session using AES and RSA

After the second session is established, we check the counters in the first session

```
ubuntu@zlin42:~$ sudo icastats -A
*** some lines not displayed ***
user: root
 function      |          # hardware       |        # software
-------------+-------------------------+-------------------------
```

 ©Copyright IBM Corporation 2017

```
                 |        ENC     CRYPT    DEC |        ENC     CRYPT    DEC
-----------------+-------------------------------+-------------------------
          SHA-1  |              422             |                0
        SHA-224  |                0             |                0
        SHA-256  |               28             |                0
        SHA-384  |                0             |                0
        SHA-512  |                0             |                0
          GHASH  |                0             |                0
          P_RNG  |                0             |                0
 DRBG-SHA-512    |              169             |                0
         RSA-ME  |                0             |                0
        RSA-CRT  |                1             |                0
        DES ECB  |        0                 0   |        0                 0
. . .
        AES CTR  |        0                 0   |        0                 0
. . .
```

Example 55: Check for increased counters for the root user

and see some increased counters (userid *root*) for SHA-1, DRBG-SHA-512 and RSA-CRT, as of the activities when the session is established (handshake), but AES CTR is still unchanged.

Further activities in the second session (like *ls* command, etc.), only affects the SHA-1 counter:

```
ubuntu@zlin42:~$ sudo icastats -A
*** some lines not displayed ***
user: root
 function        |          # hardware          |        # software
-----------------+-------------------------------+-------------------------
                 |        ENC     CRYPT    DEC |        ENC     CRYPT    DEC
-----------------+-------------------------------+-------------------------
          SHA-1  |              472             |                0
        SHA-224  |                0             |                0
        SHA-256  |               28             |                0
        SHA-384  |                0             |                0
        SHA-512  |                0             |                0
          GHASH  |                0             |                0
          P_RNG  |                0             |                0
 DRBG-SHA-512    |              169             |                0
         RSA-ME  |                0             |                0
        RSA-CRT  |                1             |                0
        DES ECB  |        0                 0   |        0                 0
. . .
        AES CTR  |        0                 0   |        0                 0
. . .
```

The AES CTR counter remains zero for SSH sessions for the same reason as for Secure Copy (see also section 5.4), as AES CTR is not supported by the engine *ibmca*. To proof, that also in the case for SSH using the cipher *aes256-ctr* the CPACF support is used, we can use the command *cpacfstats* as shown in section 5.6.

5.6 Exploiting hardware crypto support of AES and SHA without using the ibmca engine

In the previous sections OpenSSL was configured to exploit the hardware cryptography functions of the IBM z hardware and this was proven by performing various test (*openssl speed* test, *scp* and *ssh*) as well as by having a look at the IBM Cryptographic Architecture status information (with *icainfo* and

 ©Copyright IBM Corporation 2017

icastats). In this section, all the following tests are performed in an Ubuntu Linux server running directly in a LPAR (without using z/VM as an hipervisor).

```
ubuntu@zlin43:~$ openssl engine
(dynamic) Dynamic engine loading support
(ibmca) Ibmca hardware engine support
```

Example 56: Verify for available OpenSSL engines: ibmca engine is still enabled

One peculiarity of the AES and SHA implementations in OpenSSL is that the (assembly) code tries to exploit the CPACF feature even if the *ibmca* engine is not active and configured.

To demonstrate this we will disable the *ibmca* engine again temporarily with:

```
ubuntu@zlin43:~$ sudo sed -i '/openssl_conf = openssl_def/s/^/#/'
  /etc/ssl/openssl.cnf
```

Example 57: Change the line with the first occurrence of *openssl_conf = openssl_def* into a comment line to disable the ibmca engine

Verify that the *ibmca* is no longer active with:

```
ubuntu@zlin43:~$ openssl engine
(dynamic) Dynamic engine loading support
```

Example 58: Verify for available OpenSSL engines: ibmca engine is now disabled

and now observe the system counters for AES and SHA that can be displayed using the *cpacfstats* command inside of Ubuntu Server on an IBM LinuxONE or IBM Z LPAR. This is currently not possible with z/VM guests or KVM virtual machines.

For using *cpacfstats* it is required that the LPAR is setup with the option *Crypto activity counter set authorization control* marked as active. This option can be set under *Counter Facility Security Options* in the *Activation Profile*.

The s390-tools are mandatory on an Ubuntu on IBM LinuxONE or IBM Z installation and are already installed; it includes the *cpacfstats* daemon (*cpacfstatsd*) and the *cpacfstats* command:

```
ubuntu@zlin43:~$ which cpacfstats cpacfstatsd
/usr/bin/cpacfstats
/usr/sbin/cpacfstatsd
```

Example 59: cpacfstats and cpacfstatsd are available

The *cpacfstats* daemon (*cpacfstatsd*) requires root privileges and only root or members of the group *cpacfstats* are allowed to communicate with the daemon process.

Hence the following configuration, that is described in more detail in [13] and [14] is required before *cpacfstatsd* can be used.

First create a group named *cpacfstats*, in case it does not already exists:

```
ubuntu@zlin43:~$ sudo groupadd cpacfstats
```

Add all users that are allowed to run the *cpacfstats* to that group:

```
ubuntu@zlin43:~$ sudo usermod -a -G cpacfstats ubuntu
```

Example 60: Add the user *ubuntu* to *cpacfstats* group

Verify that the group modifications have taken effect with (a re-login is required, or just a *su <user>*):

```
ubuntu@zlin43:~$ groups
ubuntu adm cdrom sudo dip plugdev cpacfstats lpadmin sambashare libvirt
```

Now the *cpacfstatsd* can be started with

```
ubuntu@zlin43:~$ sudo cpacfstatsd
```

Example 61: Start the daemon cpacfstatsd

 ©Copyright IBM Corporation 2017

Verify that the daemon is properly running with:

```
ubuntu@zlin43:~$ ps aux | grep [c]pacfstats
root       3460  0.0  0.1   2808  2040 ?        Ss   10:50   0:00 cpacfstatsd
ubuntu@zlin43:~$ grep cpacfstatsd /var/log/syslog
Jan 30 10:50:31 zlin43 cpacfstatsd: cpacfstatsd: Running
```

Example 62: Check status of daemon cpacfstatsd

A simple call of the *cpacfstats* command lists the four available counters, that are all disabled by default. They can be individual enabled (AES for example with *-e aes*) or all at once with:

```
ubuntu@zlin43:~$ cpacfstats -e
 des counter: 0
 aes counter: 0
 sha counter: 0
 rng counter: 0
```

Example 63: Enable all counters of cpacfstats

To avoid any further usage of hardware crypto functions, even the ones possible used by the currently active ssh connection, either all (SSH) connections to the LPAR need to be closed or at least left unused during the test - the test itself is best performed via the console.

Alternatively the SSH connection from your workstation/client to the server that runs at the LPAR can be configured in a way that it would not use any cipher that exploits the built-in AES or SHA hardware functions.

Unfortunately for this test here, almost all cipher that are available by default use at least partly AES or SHA functions - only *chacha20-poly1305@openssh.com* does not seem to use these, but it's not available by default with the used workstation/client. Hence using the console (the *Operating System Messages* task at the HMC) looks like the easiest way for now.

Now connect to your HMC, open the *Daily* menu and start the *Operating System Messages* which opens the console of your LPAR and login with the user above, that you added to the *cpacfstats* group (see Example 60):

```
zlin43 login: ubuntu
Password: ********
```

Example 64: Login on the console

Re-run the *cpacfstats* command again:

```
ubuntu@zlin43:~$ cpacfstats
 des counter: 0
 aes counter: 55
 sha counter: 0
 rng counter: 0
```

Example 65: AES counter is increased

and in case the counters got already increased just reset them with:

```
ubuntu@zlin43:~$ cpacfstats -r
 des counter: 0
 aes counter: 0
 sha counter: 0
 rng counter: 0
```

Example 66: Reset cpacfstats counter

At this point in time you may also call the *icastats* command that displays the counters for the cryptographic functions based on the *ibmca* and *libica* layer.

 ©Copyright IBM Corporation 2017

```
ubuntu@zlin43:~$ icastats
 function      |          # hardware        |        # software
-------------+---------------------------+---------------------------
              |       ENC     CRYPT    DEC |        ENC     CRYPT    DEC
-------------+---------------------------+---------------------------
        SHA-1 |                0           |                0
. . .
      AES XTS |         0                0 |         0                0
```

Example 67: No counters of icastats are increased

In general the output should be zeros only, because we disabled the usage of *ibmca* and *libica*. But depending on the former use of that LPAR system some counter may be non-zero. In this case, reset these with:

```
ubuntu@zlin43:~$  sudo icastats -R
```

You should now have all counters resetted - the *cpacfstats* and the *icastats* counters.

Let's now perform some AES and SHA calculations by using the *openssl speed* test in the following Examples 68 and 72. Let's start with SHA:

```
ubuntu@zlin43:~$ openssl speed -evp sha256
Doing sha256 for 3s on 16 size blocks:
15020266 sha256's in 3.00s
Doing sha256 for 3s on 64 size blocks:
10077195 sha256's in 2.99s
Doing sha256 for 3s on 256 size blocks:
7846697 sha256's in 3.00s
Doing sha256 for 3s on 1024 size blocks:
4152914 sha256's in 3.00s
Doing sha256 for 3s on 8192 size blocks:
760613 sha256's in 3.00s
OpenSSL 1.0.2g  1 Mar 2016
built on: reproducible build, date unspecified
. . .
The 'numbers' are in 1000s of bytes per second processed.
type             16 bytes     64 bytes    256 bytes   1024 bytes   8192 bytes
sha256           80108.09k   215699.16k   669584.81k  1417527.98k  2076980.57k
```

Example 68: Openssl speed test with SHA-256 to perform SHA calculations

Now run the two statistic commands again. The *icastats* command does not show any changes, like expected, because we disabled the usage of *ibmca* engine:

```
ubuntu@zlin43:~$ icastats
 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
        GHASH |                0           |                0
        P_RNG |                0           |                0
 DRBG-SHA-512 |                0           |                0
       RSA-ME |                0           |                0
. . .
```

 ©Copyright IBM Corporation 2017

```
      AES XTS |          0                0 |          0                0
```

Example 69: icastats counters are all zero, including for SHA

But an increase of the *cpacfstats* counter can be observed:

```
ubuntu@zlin43:~$ cpacfstats
 des counter: 0
 aes counter: 0
 sha counter: 61464614
 rng counter: 0
```

Example 70: cpacfstats with increased counter for SHA

Hence we can conclude that the hardware assisted SHA crypto functions of CPACF were utilized even without using the *ibmca* engine.

Let's reset the *cpacfstats* counters again with:

```
ubuntu@zlin43:~$ cpacfstats -r
 des counter: 0
 aes counter: 0
 sha counter: 0
 rng counter: 0
```

Example 71: Reset cpacfstats counters

Now let's perform now some sample AES calculations with:

```
ubuntu@zlin43:~$ openssl speed -evp aes-128-cbc
Doing aes-128-cbc for 3s on 16 size blocks:
33909424 aes-128-cbc's in 2.99s
Doing aes-128-cbc for 3s on 64 size blocks:
19847182 aes-128-cbc's in 3.00s
Doing aes-128-cbc for 3s on 256 size blocks:
10848583 aes-128-cbc's in 3.00s
Doing aes-128-cbc for 3s on 1024 size blocks:
3681943 aes-128-cbc's in 3.00s
Doing aes-128-cbc for 3s on 8192 size blocks:
537585 aes-128-cbc's in 2.99s
OpenSSL 1.0.2g  1 Mar 2016
built on: reproducible build, date unspecified
. . .
The 'numbers' are in 1000s of bytes per second processed.
type             16 bytes     64 bytes    256 bytes   1024 bytes   8192 bytes
aes-128-cbc     181455.11k   423406.55k   925745.75k  1256769.88k  1472875.02k
```

Example 72: Openssl speed test to perform AES calculations

Let's run again the two statistic commands. The *icastats* command again doesn't show any changes, again expected, because we disabled the usage of the *ibmca* engine:

```
ubuntu@zlin43:~$ icastats
 function      |          # hardware       |        # software
-------------+-------------------------+-------------------------
              |       ENC     CRYPT    DEC |        ENC     CRYPT    DEC
-------------+-------------------------+-------------------------
        SHA-1 |                0           |                0
. . .
      AES ECB |          0                0 |          0                0
      AES CBC |          0                0 |          0                0
      AES OFB |          0                0 |          0                0
```

 ©Copyright IBM Corporation 2017

```
      AES CFB |          0          0 |          0          0
      AES CTR |          0          0 |          0          0
     AES CMAC |          0          0 |          0          0
      AES XTS |          0          0 |          0          0
```

Example 73: icastats counters are all zero, including for AES

The *cpacfstats* counters again indicate the usage of hardware crypto functions - this time mainly the AES counter got increased. But the *aes-128-cbc* test obviously used some SHA hardware crypto functions, too - hence the small increase of the SHA counter:

```
ubuntu@zlin43:~$ cpacfstats
 des counter: 0
 aes counter: 69385697
 sha counter: 121
 rng counter: 0
```

Example 74: cpacfstats with increased counter for AES and some small increase for SHA

Reset the *cpacfstats* counters again:

```
ubuntu@zlin43:~$ cpacfstats -r
 des counter: 0
 aes counter: 0
 sha counter: 0
 rng counter: 0
ubuntu@zlin43:~$
```

Example 75: Reset cpacfstats counters

If we finally perform some DES calculations we will see that these will not use hardware crypto functions at all and are performed in software only (this btw. also applies to RNG):

```
ubuntu@zlin43:~$ openssl speed -evp des-ede3
Doing des-ede3 for 3s on 16 size blocks:
5723613 des-ede3's in 3.00s
Doing des-ede3 for 3s on 64 size blocks:
1461167 des-ede3's in 2.99s
Doing des-ede3 for 3s on 256 size blocks:
367555 des-ede3's in 3.00s
Doing des-ede3 for 3s on 1024 size blocks:
92041 des-ede3's in 3.00s
Doing des-ede3 for 3s on 8192 size blocks:
11507 des-ede3's in 3.00s
OpenSSL 1.0.2g  1 Mar 2016
built on: reproducible build, date unspecified
...
The 'numbers' are in 1000s of bytes per second processed.
type             16 bytes     64 bytes    256 bytes   1024 bytes   8192 bytes
des-ede3         30525.94k    31275.82k    31364.69k    31416.66k    31421.78k
```

Example 76: Openssl speed test to perform DES calculations

The DES counter of *cpacfstats* didn't got increased - the SHA counter got increased due to the fact that the *des-ede3* calculation requires the execution of some SHA functions:

```
ubuntu@zlin43:~$ cpacfstats
 des counter: 0
 aes counter: 0
 sha counter: 121
 rng counter: 0
```

Example 77: cpacfstats with no increase for DES counter and some small increase for SHA

 ©Copyright IBM Corporation 2017

The *icastats* results are again all zero:

```
ubuntu@zlin43:~$ icastats
 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
        GHASH |                0            |                 0
        P_RNG |                0            |                 0
 DRBG-SHA-512 |                0            |                 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 CTR |          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 CTR |          0                0 |          0                0
    3DES CMAC |          0                0 |          0                0
      AES ECB |          0                0 |          0                0
. . .
```

Example 78: icastats counters are all zero, including for DES

Let's conclude with a positive DES test that exploits the DES hardware crypto functions again. Reset the *cpacfstats* counter(s):

```
ubuntu@zlin43:~$ cpacfstats -r
 des counter: 0
 aes counter: 0
 sha counter: 0
 rng counter: 0
ubuntu@zlin43:~$
```

Example 79: Reset cpacfstats counters

and enable the *ibmca* engine again with:

```
ubuntu@zlin42:~$ sudo sed -i '/openssl_conf = openssl_def/s/^#//'
   /etc/ssl/openssl.cnf
```

Example 80: Remove comment from the line with the first occurrence of *openssl_conf = openssl_def* in *openssl.cnf* to enable the *ibmca* engine

And verify that the *ibmca* engine is again available:

```
ubuntu@zlin43:~$ openssl engine
(dynamic) Dynamic engine loading support
(ibmca) Ibmca hardware engine support
```

Example 81: ibmca engine is enabled again

 ©Copyright IBM Corporation 2017

And rerun the *openssl speed* test for DES:

```
ubuntu@zlin43:~$ openssl speed -evp des-ede3
Doing des-ede3 for 3s on 16 size blocks:
27642588 des-ede3's in 2.99s
Doing des-ede3 for 3s on 64 size blocks:
17704877 des-ede3's in 3.00s
Doing des-ede3 for 3s on 256 size blocks:
7486675 des-ede3's in 3.00s
Doing des-ede3 for 3s on 1024 size blocks:
2265812 des-ede3's in 3.00s
Doing des-ede3 for 3s on 8192 size blocks:
299650 des-ede3's in 3.00s
OpenSSL 1.0.2g  1 Mar 2016
built on: reproducible build, date unspecified
. . .
The 'numbers' are in 1000s of bytes per second processed.
type             16 bytes     64 bytes    256 bytes   1024 bytes   8192 bytes
des-ede3        147920.20k   377704.04k   638862.93k   773397.16k   818244.27k
```

Example 82: Openssl speed test to perform DES calculations - rerun

A final verification shows that the *icastats* output changed now and shows several non-zero values, like for SHA-1, DRBG-SHA-512 and 3DES ECB:

```
ubuntu@zlin43:~$ icastats
 function      |          # hardware          |        # software
-------------+-------------------------------+-------------------------------
              |        ENC     CRYPT    DEC  |        ENC     CRYPT    DEC
-------------+-------------------------------+-------------------------------
        SHA-1 |              118              |                0
      SHA-224 |                0              |                0
      SHA-256 |                0              |                0
      SHA-384 |                0              |                0
      SHA-512 |                0              |                0
        GHASH |                0              |                0
        P_RNG |                0              |                0
 DRBG-SHA-512 |              169              |                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 CTR |          0                0   |          0                0
     DES CMAC |          0                0   |          0                0
     3DES ECB |   55399602                0   |          0                0
     3DES CBC |          0                0   |          0                0
     3DES OFB |          0                0   |          0                0
     3DES CFB |          0                0   |          0                0
     3DES CTR |          0                0   |          0                0
    3DES CMAC |          0                0   |          0                0
      AES ECB |          0                0   |          0                0
. . .
```

Example 83: icastats with non-zero counters

The *icastats* command shows the following counter values in the hardware column, means these functions were executed in hardware and will be listed by the *cpacfstats* command amongst some other counter changes:

 ©Copyright IBM Corporation 2017

```
ubuntu@zlin43:~$ cpacfstats
 des counter: 55679955
 aes counter: 0
 sha counter: 253
 rng counter: 341
```

Example 84: cpacfstats with increased counter for DES and some small increase for SHA and RNG

Leaving the prove about RNG to the reader, this section demonstrates and proves that AES and SHA hardware assisted cryptography is used even if the *ibmca* is not configured. That simplifies the exploitation of (at least a sub-set of) the hardware crypto functions. However the *ibmca* configuration is recommended to get the most out of the hardware crypto support provided by the IBM LinuxONE and IBM Z platform.

5.7 Selection of cipher and MAC for OpenSSH

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 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).

5.7.1 Using SHA with CPACF support versus MD5

OpenSSH uses hash-based Message Authentication Codes (HMAC). CPACF provides support for SHA Message Authentication Code. In a pure software environment, MD5[11] is usually faster than SHA[12], and therefore MD5 was and is still 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 (see section 5.7.2).

As already described in prior articles ([2] and [3]), from a performance perspective there is no advantage to use MD5 instead of SHA in a LinuxONE or IBM Z environment.

If you need to keep the MD5 algorithm in the list of available MACs for compatibility reasons, you need to add MD5 to the default list of algorithms and might want to place MD5 at the end of the search order (see section 5.7.3 and 5.7.4).

5.7.2 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. 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 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

[11] MD5 is a very weak hash from a security perspective.

[12] 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.

 ©Copyright IBM Corporation 2017

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 eventually also 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* or scp command. Presumably this will be the common case, and explicitly specifying a cipher or a MAC like

```
scp -c aes128-cbc -o MACs=hmac-sha2-256 testdata.txt ubuntu@localhost:/dev/null
```

or

```
ssh -c aes256-ctr -o MACs=hmac-sha2-512 ubuntu@localhost
```

is an exception.

The Ubuntu Server supports a wide range of ciphers and message authentication code algorithms. The list of available ciphers and MACs is mentioned in the man pages, or may also may be obtained using the command *ssh -Q cipher* (compare with Example 53) and the command *ssh -Q mac*. Please note, not all supported algorithms are enabled by default in the configurations of the SSHD server and SSH client.

5.7.3 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.

The list of supported ciphers and MACs for the SSHD server configuration can be found in the man page *sshd_config*. This man page contains also the list of the default ciphers. The default ciphers can also be obtained with the following command:

```
ubuntu@zlin42:~$ sshd -T 2>&1 | grep -i ciphers
ciphers chacha20-poly1305@openssh.com,
    aes128-ctr,aes192-ctr,aes256-ctr,
    aes128-gcm@openssh.com,aes256-gcm@openssh.com
```

The list of the default MACs can also be obtained with the following command:

```
ubuntu@zlin42:~$ sshd -T 2>&1 | grep -i macs
    macs umac-64-etm@openssh.com,umac-128-etm@openssh.com,
    hmac-sha2-256-etm@openssh.com,hmac-sha2-512-etm@openssh.com,
    hmac-sha1-etm@openssh.com,umac-64@openssh.com,umac-128@openssh.com,
    hmac-sha2-256,hmac-sha2-512,hmac-sha1
```

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 AES and SHA algorithms which benefit from CPACF support[13] (see Example 85). Note that a modification of the *sshd_config* file will only take effect after a restart of the SSHD daemon.

```
...
Ciphers   aes256-ctr,aes192-ctr,aes128-ctr,aes256-gcm@openssh.com,
aes128-gcm@openssh.com,aes256-cbc,aes192-cbc,aes128-cbc
```

[13] Allowing only AES and SHA algorithms might, or might not be applicable for general environments. There are regulations and other aspects to considered as well.

 ©Copyright IBM Corporation 2017

```
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
...
```

Example 85: *sshd_config* file: modification to use CPACF support

5.7.4 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[14] (see Example 86) can be modified.

```
...
#   Protocol 2
#   Cipher 3des
#   Ciphers aes128-ctr,aes192-ctr,aes256-ctr,arcfour256,arcfour128,
#       aes128-cbc,3des-cbc
#   MACs hmac-md5,hmac-sha1,umac-64@openssh.com,hmac-ripemd160
...
```

Example 86: 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 order is:

```
chacha20-poly1305@openssh.com,
aes128-ctr,aes192-ctr,aes256-ctr,
aes128-gcm@openssh.com,aes256-gcm@openssh.com,
aes128-cbc,aes192-cbc,aes256-cbc,3des-cbc
```

The default list and search order for the MACs is:

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

In Example 87, 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[15].

```
...
#   Protocol 2
#   Cipher 3des
#   Ciphers aes128-ctr,aes192-ctr,aes256-ctr,arcfour256,arcfour128,
#       aes128-cbc,3des-cbc
#   MACs hmac-md5,hmac-sha1,umac-64@openssh.com,hmac-ripemd160
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
MACs hmac-sha2-256-etm@openssh.com,hmac-sha2-512-etm@openssh.com,
hmac-sha2-256,hmac-sha2-512,hmac-sha1-etm@openssh.com,hmac-sha1,
umac-64-etm@openssh.com,umac-128-etm@openssh.com,
umac-64@openssh.com,umac-128@openssh.com
...
```

Example 87: Modified search order in *ssh_config* file to benefit from CPACF

[14] Be aware: The list of available ciphers and MACs in the SSH client configuration is not necessarily identical with the available ciphers and MACs of the SSHD server configuration.

[15] You can even more optimize the usage of hardware crypto capabilities of LinuxONE or IBM Z servers if you specify available key exchange (KEYX) algorithms that can benefit from hardware support.

 ©Copyright IBM Corporation 2017

5.8 Crypto Express support for RSA with OpenSSH

In Example 54 and 55 in section 5.5 we showed that OpenSSH can utilize RSA hardware cryptographic support from a Crypto Express feature[16]. 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.

```
ubuntu@zlin42:~$ ls -lh testdata_short
-rw-rw-r-- 1 ubuntu ubuntu 2 Apr 19 18:01 testdata_short
```

Example 88: Small file to be used by SCP

After a reset of the *icastats* counters, we use the SCP command as indicated in Example 89 multiple times.

```
ubuntu@zlin42:~$ time scp -c aes256-ctr -o HostKeyAlgorithms=ssh-rsa
   testdata_short localhost:/dev/null
ubuntu@localhost's password:
testdata_short                                    100%    2     0.0KB/s   00:00

real    0m4.066s
user    0m0.011s
sys     0m0.003s
```

Example 89: Secure Copy of a small file using RSA

Then we verify using *icastats* that RSA is really executed in the hardware (see Example 90).

```
ubuntu@zlin42:~$ icastats
 function      |          # hardware        |        # software
-------------+---------------------------+---------------------------
              |       ENC    CRYPT    DEC |        ENC    CRYPT    DEC
-------------+---------------------------+---------------------------
        SHA-1 |              484          |                0
      SHA-224 |                0          |                0
      SHA-256 |               64          |                0
      SHA-384 |                0          |                0
      SHA-512 |                0          |                0
        GHASH |                0          |                0
        P_RNG |                0          |                0
 DRBG-SHA-512 |              676          |                0
       RSA-ME |                4          |                0
      RSA-CRT |                0          |                0
. . .
```

Example 90: 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 section 5.1). Instead of

```
default_algorithms = ALL
```

we reduce the default to

```
default_algorithms = RAND,CIPHERS
```

[16] To benefit from CEX5S when using the *ibmca* engine, the CEX5S must be configured either as CEX5A or as CEX5S

 ©Copyright IBM Corporation 2017

as shown in Example 43. 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 89, 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[17] SCP request.

5.9 Apache on Ubuntu - using mod_ssl

This section provides information about how to configure an Apache web server under Ubuntu 16.04 LTS to exploit cryptographic hardware functions available with IBM Z.
The Apache interface to OpenSSL is the *mod_ssl* module. OpenSSL provides built-in CPACF-support for AES in ECB, CBC, CTR, CCM and GCM mode, SHA-1, SHA-256 and SHA-512 as well as for the latter two in their truncated versions, SHA-224 respectively SHA-384.
The *ibmca* engine is the OpenSSL interface to the *libica* library which provides CPACF-support for various ciphers, MACs and hashes, for NIST SP 800-90 compliant pseudo-random number generation as well as support for Crypto Express (CEX5S) adapters in Accelerator (CEX5A) or CCA Coprocessor (CEX5C) mode. These crypto-adapters accelerate the modular exponentiation operation that is used in the RSA, DH and DSA public-key crypto systems. If a CEX5C adapter is available, its hardware random number generator is used to seed libica's pseudo random number generation.

5.9.1 Prerequisite tasks

To install the required packages, do:

```
ubuntu@zlin42:~$ apt install apache2 openssl openssl-ibmca libica2 libica-utils
```

All packages except the apache2 package should already be present (see also Example 12).

If you have CEX5A or CEX5C adapter available, make sure that the zcrypt device driver is loaded:

```
ubuntu@zlin42:~$ modprobe ap
```

5.9.2 Configuring OpenSSL

You have to prepare your OpenSSL configuration file */etc/ssl/openssl.cnf* to enable *ibmca* engine support either with an editor, or perform the steps as shown in chapter 5 in Examples 17 to 21. If an CEX5S adapter is available for your server, we recommend, that you adapt the OpenSSL configuration to use the following as default

```
default_algorithms = RAND,RSA,DSA,DH
```

as already mentioned in section 5.1 in Example 44. The *ibmca* engine should now appear in OpenSSL's engine list as shown in Example 23.

The default_algorithms list specifies the algorithms for which the engine is used by default. In the example above we chose RAND to exploit CPACF-support for pseudo random number generation and RSA, DSA and DH to exploit CEX5A/CEX5C support for modular exponentiation. CIPHERS and DIGESTS algorithms are handled by OpenSSL's built-in CPACF-support if no engine support is requested explicitly.

Example:

```
# icastats -r
# openssl speed -evp aes-128-cbc
# icastats
# openssl speed -engine ibmca -evp aes-128-cbc
# icastats
```

[17] Using RSA acceleration support of CEX5S will have a visible effect, when multiple requests are executed in parallel.

 ©Copyright IBM Corporation 2017

With the OpenSSL configuration described above, in the first *icastats* output, the counters in the "AES CBC" line will all be zero, since CIPHERS will be handled by OpenSSL's built-in CPACF-support by default. In the second *icastas* output, you will see non-zero "AES CBC" counters, since engine support was requested explicitly.

```
# icastats -r
# openssl speed dsa
# icastats
# icastats -r
# openssl speed -engine ibmca dsa
# icastats
```

With the OpenSSL configuration described above, both *icastats* outputs will show non-zero "RSA-ME" counters, since DSA will be handled by the engine, even if not requested explicitly (the "RSA-ME" counter in fact counts the modular exponentiation operations).

5.9.3 Configuring Apache

The Apache configuration files reside in /etc/apache2.

Adjust the default HTTPS server configuration */etc/apache2/sites-available/default-ssl.conf* according your needs. For example, fill in your e-mail address:

```
ServerAdmin YOUR@MAIL.ADDRESS
```

The *mod_ssl* module configuration file is */etc/apache2/mods-available/ssl.conf*.

Add the following line to your configuration

```
SSLCryptoDevice ibmca
```

so that the HTTPS server uses the *ibmca* engine. Bear in mind that with this setting, the engine will be used for *every* algorithm, that the engine provides, the default_algorithms line in openssl.cnf plays no role here.

The cipher suites used by Apache for negotiation in SSL handshakes can be influenced by adding a list of colon-separated expressions to the SSLCipherSuite directive (see Example 91)

```
SSLCipherSuite <cipher-spec>
```

Example 91: SSLCipherSuite directive

in the *mod_ssl* configuration file.

There are also aliases for certain groups of cipher suites: For example, SSLv2, SSLv3, TLSv1 for all SSL version 2.0, 3.0 respectively TLS version 1.0 cipher suites and EXP for all export cipher suites. A complete list can be found in [15]. Cipher suites without a prefix are added to the list. Available prefixes are "+", "-" and "!". The "+" prefix pulls cipher suites to the current location. The "-" and "!" prefixes removes cipher suites from the list. Cipher suites removed by the "-" prefix can be added later again. The cipher suites are sorted from high-priority (left) to low-priority (right).

To modify and set up successively your correct *<cipher-spec>* string for the *SSLCipherSuite* directive, you can use the command

```
openssl ciphers -v '<cipher-spec>'
```

to display the resulting list of cipher suites.

By default, the clients preference is used, when choosing a cipher suite during the handshake. If you add

```
SSLHonorCipherOrder on
```

to your *mod_ssl* module configuration file, the server's preference will be used instead.

 ©Copyright IBM Corporation 2017

5.9.4 Choosing SSL/TLS cipher suites

An ideal web server would start from a "security by default"-setup, meaning that the default configuration settings would be the most secure settings possible. Because of various attacks on SSL/TLS like BREACH/CRIME, BEAST and POODLE (just to name a few) that would mean to only enable TLS 1.2 cipher suites that offer Perfect Forward Security (PFS)...

```
ubuntu@zlin42:~$ openssl ciphers -v 'ECDHE:DHE:!SSLv3'
ECDHE-RSA-AES256-GCM-SHA384 TLSv1.2 Kx=ECDH     Au=RSA   Enc=AESGCM(256) Mac=AEAD
ECDHE-ECDSA-AES256-GCM-SHA384 TLSv1.2 Kx=ECDH   Au=ECDSA Enc=AESGCM(256) Mac=AEAD
ECDHE-RSA-AES256-SHA384 TLSv1.2 Kx=ECDH     Au=RSA   Enc=AES(256)  Mac=SHA384
ECDHE-ECDSA-AES256-SHA384 TLSv1.2 Kx=ECDH   Au=ECDSA Enc=AES(256)  Mac=SHA384
ECDHE-RSA-AES128-GCM-SHA256 TLSv1.2 Kx=ECDH     Au=RSA   Enc=AESGCM(128) Mac=AEAD
ECDHE-ECDSA-AES128-GCM-SHA256 TLSv1.2 Kx=ECDH   Au=ECDSA Enc=AESGCM(128) Mac=AEAD
ECDHE-RSA-AES128-SHA256 TLSv1.2 Kx=ECDH     Au=RSA   Enc=AES(128)  Mac=SHA256
ECDHE-ECDSA-AES128-SHA256 TLSv1.2 Kx=ECDH   Au=ECDSA Enc=AES(128)  Mac=SHA256
DHE-DSS-AES256-GCM-SHA384 TLSv1.2 Kx=DH       Au=DSS   Enc=AESGCM(256) Mac=AEAD
DHE-RSA-AES256-GCM-SHA384 TLSv1.2 Kx=DH       Au=RSA   Enc=AESGCM(256) Mac=AEAD
DHE-RSA-AES256-SHA256   TLSv1.2 Kx=DH       Au=RSA   Enc=AES(256)  Mac=SHA256
DHE-DSS-AES256-SHA256   TLSv1.2 Kx=DH       Au=DSS   Enc=AES(256)  Mac=SHA256
DHE-DSS-AES128-GCM-SHA256 TLSv1.2 Kx=DH       Au=DSS   Enc=AESGCM(128) Mac=AEAD
DHE-RSA-AES128-GCM-SHA256 TLSv1.2 Kx=DH       Au=RSA   Enc=AESGCM(128) Mac=AEAD
DHE-RSA-AES128-SHA256   TLSv1.2 Kx=DH       Au=RSA   Enc=AES(128)  Mac=SHA256
DHE-DSS-AES128-SHA256   TLSv1.2 Kx=DH       Au=DSS   Enc=AES(128)  Mac=SHA256
```

... or even to remove non-AEAD cipher suites from that list.

However, a lot of older clients would be unable to connect to that web server. The problem to chose the right cipher suites for your HTTPS environment, consists essentially in finding the right balance between security and availability of your service. Furthermore, from a performance point of view, one would like to use cipher suites that enjoy hardware support, so in fact the decision becomes a trade-off between security, availability and performance.

Example:
Our strategy is to start with larger group of cipher suites. Afterwards, we disable all cipher suites that we do not trust but keep in mind that every disabled cipher suite may reduce your service's userbase. Later, we sort the list such that those cipher suites are preferred that exploit our hardware.
We start here with the group of all cipher suites that offer PFS, namely all cipher suites that use Ephemeral Diffie-Hellman keys (ECDHE/DHE).

```
ubuntu@zlin42:~$ openssl ciphers -v 'ECDHE:DHE' | grep None
```

We see that the group already excludes cipher suites without authentication, namely Anonymous (Elliptic Curve) Diffie-Hellman (ADH/AECDH).

- However, it still includes cipher suites without encryption and we remove them from our list by appending ":!eNULL".
- Old RC4 is the next candidate to be removed (":!RC4").
- Due to the newly discovered birthday attack on 64-bit block ciphers ("sweet32"), also 3DES has to be eliminated (":!3DES").

By now, our list for the *SSLCipherSuite* directive looks like shown in Example 92.

```
SSLCipherSuite DHE:ECDHE:!eNULL:!RC4:!3DES:+AES:+CAMELLIA:+SEED
```

Example 92: SSLCipherSuite directive sample

If we sort the list by availability of hardware support, we end up with this list:

 ©Copyright IBM Corporation 2017

```
ubuntu@zlin42:~$ openssl ciphers -v 'DHE:ECDHE:!eNULL:!RC4:!3DES:+AES:
   +CAMELLIA:+SEED'
DHE-DSS-AES256-GCM-SHA384 TLSv1.2 Kx=DH       Au=DSS   Enc=AESGCM(256) Mac=AEAD
DHE-RSA-AES256-GCM-SHA384 TLSv1.2 Kx=DH       Au=RSA   Enc=AESGCM(256) Mac=AEAD
DHE-RSA-AES256-SHA256     TLSv1.2 Kx=DH       Au=RSA   Enc=AES(256)  Mac=SHA256
DHE-DSS-AES256-SHA256     TLSv1.2 Kx=DH       Au=DSS   Enc=AES(256)  Mac=SHA256
DHE-RSA-AES256-SHA        SSLv3 Kx=DH         Au=RSA   Enc=AES(256)  Mac=SHA1
DHE-DSS-AES256-SHA        SSLv3 Kx=DH         Au=DSS   Enc=AES(256)  Mac=SHA1
DHE-DSS-AES128-GCM-SHA256 TLSv1.2 Kx=DH       Au=DSS   Enc=AESGCM(128) Mac=AEAD
DHE-RSA-AES128-GCM-SHA256 TLSv1.2 Kx=DH       Au=RSA   Enc=AESGCM(128) Mac=AEAD
DHE-RSA-AES128-SHA256     TLSv1.2 Kx=DH       Au=RSA   Enc=AES(128)  Mac=SHA256
DHE-DSS-AES128-SHA256     TLSv1.2 Kx=DH       Au=DSS   Enc=AES(128)  Mac=SHA256
DHE-RSA-AES128-SHA        SSLv3 Kx=DH         Au=RSA   Enc=AES(128)  Mac=SHA1
DHE-DSS-AES128-SHA        SSLv3 Kx=DH         Au=DSS   Enc=AES(128)  Mac=SHA1
ECDHE-RSA-AES256-GCM-SHA384 TLSv1.2 Kx=ECDH   Au=RSA   Enc=AESGCM(256) Mac=AEAD
ECDHE-ECDSA-AES256-GCM-SHA384 TLSv1.2 Kx=ECDH Au=ECDSA Enc=AESGCM(256) Mac=AEAD
ECDHE-RSA-AES256-SHA384 TLSv1.2 Kx=ECDH       Au=RSA   Enc=AES(256)  Mac=SHA384
ECDHE-ECDSA-AES256-SHA384 TLSv1.2 Kx=ECDH     Au=ECDSA Enc=AES(256)  Mac=SHA384
ECDHE-RSA-AES256-SHA      SSLv3 Kx=ECDH       Au=RSA   Enc=AES(256)  Mac=SHA1
ECDHE-ECDSA-AES256-SHA    SSLv3 Kx=ECDH       Au=ECDSA Enc=AES(256)  Mac=SHA1
ECDHE-RSA-AES128-GCM-SHA256 TLSv1.2 Kx=ECDH   Au=RSA   Enc=AESGCM(128) Mac=AEAD
ECDHE-ECDSA-AES128-GCM-SHA256 TLSv1.2 Kx=ECDH Au=ECDSA Enc=AESGCM(128) Mac=AEAD
ECDHE-RSA-AES128-SHA256 TLSv1.2 Kx=ECDH       Au=RSA   Enc=AES(128)  Mac=SHA256
ECDHE-ECDSA-AES128-SHA256 TLSv1.2 Kx=ECDH     Au=ECDSA Enc=AES(128)  Mac=SHA256
ECDHE-RSA-AES128-SHA      SSLv3 Kx=ECDH       Au=RSA   Enc=AES(128)  Mac=SHA1
ECDHE-ECDSA-AES128-SHA    SSLv3 Kx=ECDH       Au=ECDSA Enc=AES(128)  Mac=SHA1
DHE-RSA-CAMELLIA256-SHA   SSLv3 Kx=DH         Au=RSA   Enc=Camellia(256) Mac=SHA1
DHE-DSS-CAMELLIA256-SHA   SSLv3 Kx=DH         Au=DSS   Enc=Camellia(256) Mac=SHA1
DHE-RSA-CAMELLIA128-SHA   SSLv3 Kx=DH         Au=RSA   Enc=Camellia(128) Mac=SHA1
DHE-DSS-CAMELLIA128-SHA   SSLv3 Kx=DH         Au=DSS   Enc=Camellia(128) Mac=SHA1
DHE-RSA-SEED-SHA          SSLv3 Kx=DH         Au=RSA   Enc=SEED(128)  Mac=SHA1
DHE-DSS-SEED-SHA          SSLv3 Kx=DH         Au=DSS   Enc=SEED(128)  Mac=SHA1
```

All above cipher suites offer PFS and the employed algorithms are reasonably safe. Moreover, the cipher suites that exploit hardware support are preferred.

A note on SHA1: SHA1 is deprecated for signing TLS certificates and browsers will block such certificates (see [16]). However, using SHA1 as a HMAC was still considered to be safe because the security assumptions for certificate signatures and HMAC are different. Recently we observed the first practical collision attack ("SHAttered") indicating that the lifetime of SHA1 comes to an end. If possible, SHA1 should also be deleted from the above list. In this case, the first list is obtained, featuring only TLS1.2 cipher suites.

5.9.5 Starting the web server

To enable HTTPS and *mod_ssl*, do:

```
ubuntu@zlin42:~$ a2ensite default-ssl
ubuntu@zlin42:~$ a2enmod ssl
```

To start the web server, do:

```
ubuntu@zlin42:~$ systemctl restart apache2.service
```

Check if any errors occurred:

```
ubuntu@zlin42:~$ systemctl status apache2.service
ubuntu@zlin42:~$ cat /var/log/apache2/error.log
```

 ©Copyright IBM Corporation 2017

Open https://FQDN in your browser, where FQDN is to be replaced with your server's fully qualified domain name. The *Apache2 Ubuntu Default Page* (/var/www/html/index.html) should show up.
You can also use OpenSSL's s_client to connect to your server, requesting a specific cipher suite via the *-cipher* option:

```
openssl s_client -connect FQDN:443 -cipher 'DHE-RSA-AES256-SHA' -debug
```

6 Configuring PKCS#11 environment

The PKCS#11 interface is the second method that applications request cryptographic services in a standardized manner. The openCryptoki package is the open source implementation of IBM for the PKCS#11 interface to provide cryptographic hardware devices that can manage and store user keys on PKCS#11 devices (see also [17]). openCryptoki consists of a slot manager and an API for slot token dynamic link libraries (STDLLs). The slot manager runs as a daemon to control token slots provided to applications. Managed devices store tokens in the slot manager database. Multiple slots with a token can be configured within one environment (see Figure 3). openCryptoki supports different tokens, which support different sets of cryptographic algorithms and different ways of operating (encryption in pure software, encryption with hardware support, clear key only encryption, secure key encryption,...). After installation of openCryptoki, the tokens to be used have to be configured. The access to the administrative functions for the PKCS#11 device is secured by the SO PIN (security officer PIN). To access the token stored in the PKCS#11 device slot database a user PIN is required. Note, in the following examples we use any arbitrary values for the PINs. In a production environment, you would carefully choose your values according to existing security policies.

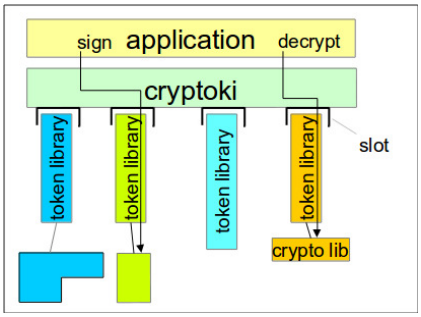

Figure 3: PKCS#11 architecture

Components provided by openCryptoki include:

- Slot manager daemon (/usr/sbin/pkcsslotd)
- Slot manager daemon service control script (/etc/init.d/pkcsslotd)
- APIs to the STDLLs (/usr/lib/opencryptoki/libopencryptoki.so)
- Configuration utilities:
  - /usr/sbin/pkcsconf
  - /usr/sbin/pkcscca
  - /usr/sbin/pkcsicsf
- STDLLs plugins to the cryptographic adapters: /usr/lib/opencryptoki/stdll/

6.1 Installation and preparation of openCryptoki

To install an openCryptoki environment (in our project, we only look at clear key encryption), you need to install two packages[18]:

```
sudo apt-get install libtspi1 opencryptoki
```

Example 93: Install packages for openCryptoki

[18]The package libtsp1 is conjunction with Trusted Computing Group's Software Stack (TSS) and using TPM hardware. As of today it has to be installed prior of installing openCryptoki even it is not used in our environment.

 ©Copyright IBM Corporation 2017

The support for clear key operations for the software-only token and the IBM ICA token for using CPACF and CEX5A or CEX5C feature is already contained.

To configure openCryptoki, proceed with the following steps

- Ensure zcrypt driver is loaded
- Check group membership for users
- Enable pkcsslot daemon
- Start pkcsslot daemon
- Configure the tokens using pkcsconf

Before you start to configure the tokens, the zcrypt device driver must be loaded into the kernel. Please verify (as shown in Example 29 and 31) that the device driver is already loaded.

Users who should be allowed to access the openCryptoki library must be a member of the *pkcs11* group. After verification, that the *pkcs11* group is already existing, add the userids to *pkcs11* group:

```
ubuntu@zlin42:~/opencryptoki/testcases/crypto$ sudo usermod -aG pkcs11 gnirss
ubuntu@zlin42:~/opencryptoki/testcases/crypto$ sudo usermod -aG pkcs11 tschoeke
ubuntu@zlin42:~/opencryptoki/testcases/crypto$ sudo usermod -aG pkcs11 ubuntu
```

Example 94: Add users to pkcs11 group

Note, that group membership becomes active when a session is opened. It might be necessary for a user to logout and login again. In our environment, now we have the following members defined in the *pkcs11* group:

```
grep pkcs11 /etc/group
pkcs11:x:118:root,gnirss,tschoeke,ubuntu
```

Example 95: Members of pkcs11 group as defined in /etc/group

Now, enable pkcsslot daemon to restart after a reboot of the system:

```
ubuntu@zlin42:~$ sudo systemctl enable pkcsslotd.service
Synchronizing state of pkcsslotd.service with SysV init
with /lib/systemd/systemd-sysv-install...
Executing /lib/systemd/systemd-sysv-install enable pkcsslotd
```

Start slot daemon:

```
ubuntu@zlin42:~$ sudo systemctl start pkcsslotd.service
```

To display the current token information, use the *-t* option with *pkcsconf*:

```
ubuntu@zlin42:~$ pkcsconf -t
Token #1 Info:
        Label: IBM ICA  PKCS #11
        Manufacturer: IBM Corp.
        Model: IBM ICA
        Serial Number: 123
        Flags: 0x880045 (RNG|LOGIN_REQUIRED|CLOCK_ON_TOKEN|USER_PIN_TO_BE_CHANGED|
   SO_PIN_TO_BE_CHANGED)
        Sessions: 0/18446744073709551614
        R/W Sessions: 18446744073709551615/18446744073709551614
        PIN Length: 4-8
        Public Memory: 0xFFFFFFFFFFFFFFFF/0xFFFFFFFFFFFFFFFF
        Private Memory: 0xFFFFFFFFFFFFFFFF/0xFFFFFFFFFFFFFFFF
        Hardware Version: 1.0
        Firmware Version: 1.0
```

 ©Copyright IBM Corporation 2017

```
        Time: 14:32:19
Token #3 Info:
        Label: IBM OS PKCS#11
        Manufacturer: IBM Corp.
        Model: IBM SoftTok
        Serial Number: 123
        Flags: 0x880045 (RNG|LOGIN_REQUIRED|CLOCK_ON_TOKEN|USER_PIN_TO_BE_CHANGED|
   SO_PIN_TO_BE_CHANGED)
        Sessions: 0/18446744073709551614
        R/W Sessions: 18446744073709551615/18446744073709551614
        PIN Length: 4-8
        Public Memory: 0xFFFFFFFFFFFFFFFF/0xFFFFFFFFFFFFFFFF
        Private Memory: 0xFFFFFFFFFFFFFFFF/0xFFFFFFFFFFFFFFFF
        Hardware Version: 1.0
        Firmware Version: 1.0
        Time: 14:32:19
```

Example 96: Display PKCS#11 configuration

Please note: IBM ICA PKCS #11 is the default token label that is changed at token initialization.

To display PKCS#11 information, use the *-i* option:

```
ubuntu@zlin42:~$ pkcsconf -i
PKCS#11 Info
        Version 2.20
        Manufacturer: IBM
        Flags: 0x0
        Library Description: Meta PKCS11 LIBRARY
        Library Version 3.4
```

Example 97: Display PKCS#11 information

To display slot information, use the *-s* option:

```
ubuntu@zlin42:~$ pkcsconf -s
Slot #1 Info
        Description: Linux
        Manufacturer: IBM
        Flags: 0x1 (TOKEN_PRESENT)
        Hardware Version: 0.0
        Firmware Version: 0.0
Slot #3 Info
        Description: Linux
        Manufacturer: IBM
        Flags: 0x1 (TOKEN_PRESENT)
        Hardware Version: 0.0
        Firmware Version: 0.0
```

Example 98: Display slot information

The slot number can be found in /etc/opencryptoki/opencryptoki.conf

```
version opencryptoki-3.1
# The following defaults are defined:
#       hwversion = 0.0
#       firmwareversion = 0.0
#       description = Linux
#       manufacturer = IBM
#
# The slot definitions below may be overriden and/or customized.
```

 ©Copyright IBM Corporation 2017

```
# For example:
#       slot 0
#       {
#          stdll = libpkcs11_cca.so
#          description = "OCK CCA Token"
#          manufacturer = "MyCompany Inc."
#          hwversion = 2.32
#          firmwareversion = 1.0
#       }
#
# See man(5) opencryptoki.conf for further information.
#
slot 0
{
stdll = libpkcs11_tpm.so
}
slot 1
{
stdll = libpkcs11_ica.so
}
slot 2
{
stdll = libpkcs11_cca.so
}
slot 3
{
stdll = libpkcs11_sw.so
}
slot 4
{
stdll = libpkcs11_ep11.so
confname = ep11tok.conf
}
```

Example 99: Slots numbers as of default

In this paper we only consider the tokens, which can be used for clear-key operation:

- ICA token, which can use hardware acceleration
- sw token, which executes encryption request in pure software.

6.1.1 Configuration of the openCryptoki ICA token

Now we configure the ICA token, which allows access to the available hardware support for clear key encryption on Linux for z Systems or LinuxONE platform. As shown in Example 96 and Example 99 we use slot 1 for the ICA token. In Ubuntu Server 16.04 LTS the slot configuration for the ICA token is stored in /var/lib/opencryptoki/lite/. To configure the PKCS#11 device, the token label must be initialized before it can be used. To initialize the token label (replacing the default label), use the *-I* option and specify the slot number using the *-c* option. We use for our example here *myicatoken* as token label. When prompted, provide the default SO PIN (87654321):

```
ubuntu@zlin42:~$ pkcsconf -I -c 1
Enter the SO PIN: 87654321
Enter a unique token label: myicatoken
```

Example 100: Initialization of ICA token (slot 1)

To check that the label has changed, use the *-t* option:

 ©Copyright IBM Corporation 2017

```
ubuntu@zlin42:~$ pkcsconf -t
Token #1 Info:
        Label: myicatoken
        Manufacturer: IBM Corp.
        Model: IBM ICA
        Serial Number: 123
        Flags: 0x880445 (RNG|LOGIN_REQUIRED|CLOCK_ON_TOKEN|TOKEN_INITIALIZED|
   USER_PIN_TO_BE_CHANGED|SO_PIN_TO_BE_CHANGED)
        Sessions: 0/18446744073709551614
        R/W Sessions: 18446744073709551615/18446744073709551614
        PIN Length: 4-8
        Public Memory: 0xFFFFFFFFFFFFFFFF/0xFFFFFFFFFFFFFFFF
        Private Memory: 0xFFFFFFFFFFFFFFFF/0xFFFFFFFFFFFFFFFF
        Hardware Version: 1.0
        Firmware Version: 1.0
        Time: 14:46:22
. . .
```

Example 101: Verify changed token label

The token label (in Example 101 it is *myicatoken*) identifies the cryptographic token.

It is good security practice to set the SO PIN to a different private non-default value. The SO PIN secures access to the administrative functions for the PKCS#11 device. Now change the SO PIN, in our example we use 13243546 as new SO PIN, using the *-P* option:

```
ubuntu@zlin42:~$ pkcsconf -P -c 1
Enter the SO PIN: 87654321
Enter the new SO PIN: 13243546
Re-enter the new SO PIN: 13243546
```

Example 102: Change the SO PIN for slot 1

As next step, the user PIN is set by the security officer. The user PIN (length is between 4 to 8 characters) secures access to the token stored in the PKCS#11 device slot database. To access the token, users must provide the user PIN[19]. To initialize a user PIN use the *-u* option:

```
ubuntu@zlin42:~$ pkcsconf -u -c 1
Enter the SO PIN: 13243546
Enter the new user PIN: 11111111
Re-enter the new user PIN: 11111111
```

Example 103: Initialize the user PIN for slot 1

To ensure the SO has no access to the token, you should change the user PIN as soon as a user is granted access. To change the user PIN use the *-p* option, in our example we use as new user PIN 87654321 for slot 1:

```
ubuntu@zlin42:~$ pkcsconf -p -c  1
Enter user PIN: 11111111
Enter the new user PIN: 87654321
Re-enter the new user PIN: 87654321
```

Example 104: Set a user PIN for slot 1

Important: Avoid the user PIN 12345678. There is a hard coded check in openCryptoki that will fail requests with that PIN (0xA1 CKR_PIN_INVALID). The PKCS#11 device in slot 1 (using hardware support of CPACF or CEX5S) is now configured to store and manage the keys for an application such as IBM Security Access Manager, IBM HTTP Server, WebSphere® MQ, or other applications using PKCS#11 interface for encryption tasks.

[19] It is good security practice, that the user PIN is different from the SO PIN.

 ©Copyright IBM Corporation 2017

6.1.2 Configuration of the openCryptoki software token

In this step we configure the token for encryption without hardware acceleration (i.e. software only). This allows us to demonstrate different behaviour. The token for software only encryption resides in our example in slot 3. In Ubuntu Server 16.04 LTS the slot configuration for the ICA token is stored in /var/lib/opencryptoki/swtok/. To configure the software token, the same steps as already performed in section 6.1.1 are necessary.

Change the token label. We use for our example *myswtoken* as token label. When prompted, provide the default SO PIN (87654321):

```
ubuntu@zlin42:~$ pkcsconf -I -c 3
Enter the SO PIN: 87654321
Enter a unique token label: myswtoken
```

Example 105: Initialization of software token (slot 3)

Change[20] the SO PIN for slot 3. We use here 13243546.

```
ubuntu@zlin42:~$ pkcsconf -P -c 3
Enter the SO PIN: 87654321
Enter the new SO PIN: 13243546
Re-enter the new SO PIN: 13243546
```

Example 106: Change the SO PIN for slot 3

Initialize user PIN to any value, like 11111111, as we will change it afterwards.

```
ubuntu@zlin42:~$ pkcsconf -u -c 3
Enter the SO PIN: 13243546
Enter the new user PIN: 11111111
Re-enter the new user PIN: 11111111
```

Example 107: Initialize the user PIN for slot 3

Change[21] the user PIN for slot 3. In our example we us 76543210.

```
ubuntu@zlin42:~$ pkcsconf -p -c 3
Enter user PIN: 11111111
Enter the new user PIN: 76543210
Re-enter the new user PIN: 76543210
```

Example 108: Set a user PIN for slot 3

6.2 Verify the configuration of openCryptoki

At first check the configuration can be done with the *-t* option of *pkcsconf*:

```
ubuntu@zlin42:~$ pkcsconf -t
Token #1 Info:
        Label: myicatoken
        Manufacturer: IBM Corp.
        Model: IBM ICA
        Serial Number: 123
        Flags: 0x44D (RNG|LOGIN_REQUIRED|USER_PIN_INITIALIZED|CLOCK_ON_TOKEN|
   TOKEN_INITIALIZED)
        Sessions: 0/18446744073709551614
        R/W Sessions: 18446744073709551615/18446744073709551614
```

[20] For simplification in our test environment, we use the same SO PIN for slot 1 and for slot 3. In a production environment, different SO PINs might be adequate.

[21] It is good security practice to have different user PINs for different tokens.

 ©Copyright IBM Corporation 2017

```
        PIN Length: 4-8
        Public Memory: 0xFFFFFFFFFFFFFFFF/0xFFFFFFFFFFFFFFFF
        Private Memory: 0xFFFFFFFFFFFFFFFF/0xFFFFFFFFFFFFFFFF
        Hardware Version: 1.0
        Firmware Version: 1.0
        Time: 00:55:16
Token #3 Info:
        Label: myswtoken
        Manufacturer: IBM Corp.
        Model: IBM SoftTok
        Serial Number: 123
        Flags: 0x44D (RNG|LOGIN_REQUIRED|USER_PIN_INITIALIZED|CLOCK_ON_TOKEN|
   TOKEN_INITIALIZED)
        Sessions: 0/18446744073709551614
        R/W Sessions: 18446744073709551615/18446744073709551614
        PIN Length: 4-8
        Public Memory: 0xFFFFFFFFFFFFFFFF/0xFFFFFFFFFFFFFFFF
        Private Memory: 0xFFFFFFFFFFFFFFFF/0xFFFFFFFFFFFFFFFF
        Hardware Version: 1.0
        Firmware Version: 1.0
        Time: 00:55:16
```

Example 109: Display PKCS#11 configuration

We compare Example 109 with Example 96 and see, that the *Flags* have changed to 0x44D. Now *SO_PIN_TO_BE_CHANGED* is not shown, and we see *TOKEN_INITIALIZED*. This is an indication, that our environment is now ready.

To verify the installation and configuration of the PKCS#11 (i.e. openCryptoki) environment, you can use the test programs provided by the openCryptoki project. The test utilities are not part of the delivery of the distributor (Canonical®). You can download the source of the complete openCryptoki package from SourceForge and create (build) the test programs by yourself and verify your environment to check, whether hardware support of IBM Z or LinuxONE servers are used to accelerate encryption workload.

As the openCryptoki run-time code is already installed, you need only to build the test programs out of the openCryptoki source code.

To build the test utilities, you need a development environment. Verify, whether your environment is already capable to be used to create the utilities, or whether you still need to install the appropriate packages:

```
apt-get install build-essential automake autoconf libtool expect libssl-dev
```

Example 110: Install packages for test environment

From SourceForge https://sourceforge.net/projects/opencryptoki/ download the openCryptoki package and extract the archive.

```
ubuntu@zlin42:~$ tar xvfz opencryptoki-3.6.1.tgz
```

Example 111: Unpack the openCryptoki package to get access to the source code

Change to the just created directory *opencryptoki* and invoke bootstrap.sh

```
cd opencryptoki
./bootstrap.sh
```

Then prepare the environment for the compilation

```
./configure --enable-testcases
```

 ©Copyright IBM Corporation 2017

As we have already installed the openCryptoki package (see Example 93) for runtime (without the executables for the test programs), we only have to compile the test programs. We do not need to compile the complete openCryptoki package, therefore we change to the subdirectory *testcases* and compile the appropriate programs

```
cd testcases
make
```

To execute the tests, change to the subdirectory *crypto*

```
cd crypto
```

Before we execute some tests, we disable dynamic engine support of *ibmca* in our OpenSSL environment[22] to avoid that our activities inside the OpenSSH terminal session increases the counters of the *icastats* output. Here, we are only interested in an increase of the counters based on activities using the openCryptoki interface and not the OpenSSL interface.

To access the openCryptoki environment, the program needs access to the openCryptoki libraries. Therefore the user must be in the group *pkcs11* and the user PIN must be available for the execution of the test program. In general, the user PIN is already built-in during compilation[23], or it is provided as user-input, or read from an application profile, or it is available via an environment variable. The test programs provided here by openCryptoki expect the user PIN via the environment variable *PKCS11_USER_PIN*.

In a first step we test encryption with openCryptoki using the software token, which resides in slot 3. Therefore we initialize the environment variable with the user PIN for the token in slot 3 (as specified in Example 108).

```
export PKCS11_USER_PIN="76543210"
```

Example 112: Provide the user PIN for the software token in slot 3

and we reset all the *icastats* counters.

```
ubuntu@zlin42:~/opencryptoki/testcases/crypto$ icastats -r
```

Now we invoke the test programs to execute AES, TDES, and RSA encryption. For simplification, we redirect the output into files.

```
ubuntu@zlin42:~/opencryptoki/testcases/crypto$ ./aes_tests -slot 3
>aes_tests_sw_out.txt 2>&1
ubuntu@zlin42:~/opencryptoki/testcases/crypto$ ./des3_tests -slot 3
>des3_tests_sw_out.txt 2>&1
ubuntu@zlin42:~/opencryptoki/testcases/crypto$ ./rsa_tests -slot 3
>rsa_tests_sw_out.txt 2>&1
```

Example 113: Perform some crypto tests using sw token in slot 3

After execution, check the result in the output files. If you find errors with the return code *rc = CKR_PIN_INCORRECT*, then you have set a wrong user PIN in the environment variable. If you find errors with the return code *rc = CKR_PIN_LOCKED*, then you have locked the user PIN and access has been revoked, probably by too many unsuccessful accesses to the token using the wrong PIN while testing. In addition, you can recognize this case also with the help of the command *pkcsconf -t* the *Flags* line of the used token will no more contain *0x44D* as shown in Example 109, but a different value and an appropriate textual indication (see Example 114).

```
Flags: 0x4044D (RNG|LOGIN_REQUIRED|USER_PIN_INITIALIZED|CLOCK_ON_TOKEN|
    TOKEN_INITIALIZED|USER_PIN_LOCKED)
```

Example 114: Token with a locked user PIN

[22] To disable dynamic engine support we simply remove (uncomment) the line *openssl_conf = openssl_def* at the top of the OpenSSL configuration file shown in Example 22.

[23] This way provides very low flexibility and might be really an exception.

 ©Copyright IBM Corporation 2017

To enable usage of the token again, the user PIN must be set (may be with the help of the SO) as indicated in Examples 103-104 or in Examples 107-108.

In the output files, you should find *Using slot #3* and some success messages, like shown in Example 115:

```
Using slot #3...

With option: nostop: 0
------
* TESTSUITE do_EncryptAES BEGIN AES_ECB Encryption.
------
* TESTCASE do_EncryptAES BEGIN AES_ECB Encryption with published test vector 0.
* TESTCASE do_EncryptAES PASS (elapsed time 0s 21us) AES_ECB Encryption with
test vector 0 passed.

------
* TESTCASE do_EncryptAES BEGIN AES_ECB Encryption with published test vector 1.
* TESTCASE do_EncryptAES PASS (elapsed time 0s 5us) AES_ECB Encryption with
test vector 1 passed.
. . .
```

Example 115: Successful execution of test using sw token in slot 3

Now we check using the *icastats* command, whether the *icastats* counters have increased during test execution. We see that only the *DRBG-SHA-512* counter[24] has an increased value.

```
ubuntu@zlin42:~/opencryptoki/testcases/crypto$ icastats
 function      |          # hardware       |       # software
---------------+-------------------------+-------------------------
               |       ENC     CRYPT   DEC |        ENC     CRYPT   DEC
---------------+-------------------------+-------------------------
        SHA-1  |                0          |                0
. . .
        P_RNG  |                0          |                0
 DRBG-SHA-512  |              676          |                0
       RSA-ME  |                0          |                0
. . .
      AES XTS  |         0               0 |         0               0
```

Example 116: No increased counters for AES DES or RSA during encryption using sw token in slot 3

In the second step we test encryption with openCryptoki using the ICA token, which resides in slot 1. Using the ICA token, encryption requests are enabled to benefit from the CPACF and CEX5S feature. Therefore we initialize the environment variable with the user PIN for the token in slot 1 (as specified in Example 104).

```
export PKC11_USER_PIN="87654321"
```

Example 117: Provide user PIN for ICA token via environment variable to the test programs

and again we reset all the *icastats* counters.

```
ubuntu@zlin42:~/opencryptoki/testcases/crypto$ icastats -r
```

Now we invoke the test programs to execute AES, TDES, and RSA encryption using the ICA token in slot 1. For simplification, we redirect the output into files.

[24] These are calls of the Deterministic Random Bit Generators (i.e. pseudo random). Random is used for any purpose like keygen, seeds for openssl, and more, so it is difficult to retrace this counter in details. Here it is important the the counter is increased and that it is performed with CPACF support.

 ©Copyright IBM Corporation 2017

```
ubuntu@zlin42:~/opencryptoki/testcases/crypto$ ./aes_tests -slot 1
>aes_tests_ica_out.txt 2>&1
ubuntu@zlin42:~/opencryptoki/testcases/crypto$ ./des3_tests -slot 1
>des3_tests_ica_out.txt 2>&1
ubuntu@zlin42:~/opencryptoki/testcases/crypto$ ./rsa_tests -slot 1
>rsa_tests_ica_out.txt 2>&1
```

Example 118: Perform some tests using ICA token in slot 1

After execution, check the result in the output files. In the output files, you should find the *Using slot #1* and some success messages, like already shown in Example 115. We expect increased counters for AES, DES and RSA, as we used the ICA token and we can confirm with the *icastats* command that hardware support for encryption has been used (see Example 119).

```
ubuntu@zlin42:~/opencryptoki/testcases/crypto$ icastats
 function      |          # hardware          |        # software
-------------+-------------------------------+-------------------------------
              |       ENC     CRYPT    DEC   |        ENC     CRYPT    DEC
-------------+-------------------------------+-------------------------------
        SHA-1 |              420              |                0
      SHA-224 |                0              |                0
      SHA-256 |              247              |                0
      SHA-384 |              139              |                0
      SHA-512 |              105              |                0
        GHASH |              104              |                0
        P_RNG |                0              |                0
 DRBG-SHA-512 |            19509              |                0
       RSA-ME |              436              |                0
      RSA-CRT |              415              |                0
      DES ECB |        30               30    |        0               0
      DES CBC |         0                0    |        0               0
      DES OFB |         0                0    |        0               0
      DES CFB |         0                0    |        0               0
      DES CTR |         0                0    |        0               0
     DES CMAC |         0                0    |        0               0
     3DES ECB |       565              565    |        0               0
     3DES CBC |      1431             1150    |        0               0
     3DES OFB |        32               32    |        0               0
     3DES CFB |        53               53    |        0               0
     3DES CTR |         0                0    |        0               0
    3DES CMAC |         0                0    |        0               0
      AES ECB |       913              853    |        0               0
      AES CBC |      1568             1565    |        0               0
      AES OFB |        11               11    |        0               0
      AES CFB |        20               20    |        0               0
      AES CTR |       871              787    |        0               0
     AES CMAC |         0                0    |        0               0
      AES XTS |         0                0    |        0               0
```

Example 119: Increased counters for AES DES or RSA during encryption using ICA token in slot 1

In our environment we can now demonstrate, that depending on the specified token for encryption work, we can benefit from hardware encryption support from a Linux for z Systems or LinuxONE environment.

 ©Copyright IBM Corporation 2017

6.3 Apache on Ubuntu - using mod_nss

This section provides information about how to configure an Apache web server under Ubuntu 16.04 LTS to exploit cryptographic hardware functions available with IBM Z.

The Apache interface to OpenSSL is the *mod_nss* module. *mod_nss* uses the PKCS#11 interface to perform cryptographic operations.

How to configure Apache using PKCS#11 interface (openCryptoki) to benefit from IBM Z hardware acceleration for cryptographic operations, please refer to [18].

6.4 Using IBM Java with hardware cryptographic support on Ubuntu

Application workloads on IBM Z are running within an enterprise environments, where data protection, integrity and confidentiality requirements are crucial. Java is a widely used language for many application types which results in a need for efficiency in execution.

In this paper we focus on IBM Java, because as of today, it is the only available Java which is optimized and supported for the IBM Z platform[25] [26]. It provides access to hardware acceleration of cryptographic functions as well as secure key cryptographic operations using Crypto Express features in a Linux for z Systems environment.

Using the Java Cryptography Architecture (JCA) along with the Java Cryptography Extension (JCE) Application Programming Interface (API) and the appropriate Java provider (see Figure 4), Java applications can get access to the IBM Z cryptographic hardware features (see [19]). JCA is a plug-in framework that supports registration of multiple providers of cryptographic functions: It provides API's for applications to check the available providers for specific services. A provider supplies the implementation of a set of security API cryptographic features that are advertised for specific cryptographic algorithms. This lets a program use cryptographic functions from any of the installed providers that support the required function. The provider to be used can either be selected explicitly by the program or automatically, according to a priority assigned to the providers when configured (see Figure 5 and Example 129 and 140). The IBMJCE provider is a implementation of such a JCE (note also section 6.4.5) to expand the functionalities of IBM Java. Alternatively, the IBMPKCS11Impl provider is an implementation of JCE that calls a function from a library implementation of the PKCS#11 standard (openCryptoki) and the hardware that supports this standard.

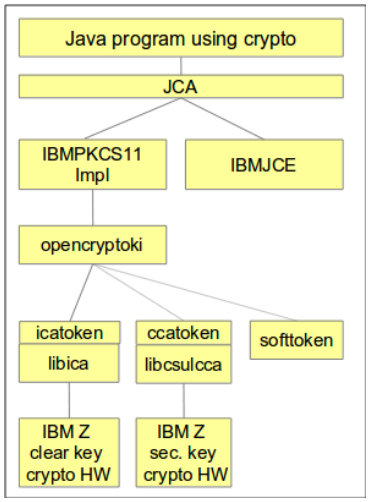

Figure 4: JCA architecture

The software stack required for Java applications to exploit IBM Z cryptographic hardware acceleration in LinuxONE or Linux for z Systems consists of three layers:

- The JCA layer in Java
- The PKCS#11 layer provided by openCryptoki
- IBM Z platform-specific libraries accessing the cryptographic hardware features and functions

[25] Oracle Java is not supported on IBM Z and LinuxONE servers while other JAVA variations often lack some functionalities.

[26] The IBM implementation of the Java platform is based upon the standrad Java Technology developed by Oracle Corporation. IBM supplies two installable packages for multiple hardware platforms: the Software Developers Kit (SDK) and the Java runtime environment. The "IBM SDK, Java Technology Edition" is fully compatible with the Oracle Platform Java Standard Edition (Java SE) application programming interfaces (APIs).

 ©Copyright IBM Corporation 2017

6.4.1 Installation of IBM Java on Ubuntu

There are two ways to install IBM Java onto your Ubuntu server:

- Download and install from developerWorks® site
- Download and install from Canonical Partner Archive

The current Web Site for downloading IBM Java 8
https://developer.ibm.com/javasdk/downloads/sdk8/
and for IBM Java 7
https://developer.ibm.com/javasdk/downloads/sdk7r1/.

Each website hosts the Java Development Kit (JDK) and Java Runtime Environment (JRE) as download only or as installable packages. The installation file consists of a shell script at the beginning followed by the package data. The downloaded binary just needs to be executed to start the installation. The process is interactive in text-mode.

The IBM Java package is also available via a Canonical Partner Archive[27]. The Canonical Partner Archive is usually already listed in the file /etc/apt/sources.list, but disabled (commented). In order to enable it, you may uncomment two lines specifying the Canonical Partner Archive with single comment character '#'. Mind 'xenial' is the release in our sample, your release might be different.

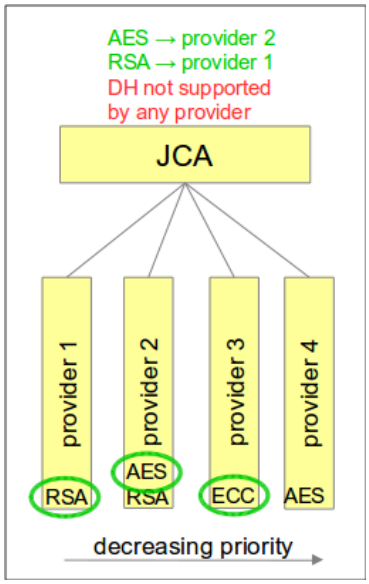

Figure 5: Selection of algorithms out of multiple providers

```
## Uncomment the following two lines to add software from Canonical's
## 'partner' repository.
## This software is not part of Ubuntu, but is offered by Canonical and the
## respective vendors as a service to Ubuntu users.
# deb http://archive.canonical.com/ubuntu xenial partner
# deb-src http://archive.canonical.com/ubuntu xenial partner
```

Example 120: Canonical Partner Archive

If the Canonical Partner Archive is not listed yet, you need to add these two lines to /etc/apt/-sources.list file. Next, you need to identify the IBM Java package and install it:

```
sudo apt install ibm-java80-jdk ibm-java80-jre
. . .
```

Example 121: Java Installation

A detailed description how to proceed can be found in [20].

6.4.2 Enable IBM Java for using strong encryption

By default, the IBM SDK, on all platforms, provides strong but limited jurisdiction policy files. However, to increase the level of security , Java Cryptography Extension (JCE) Unlimited Strength Policy Files are necessary to overcome limitations in the length of keys (see also [21]). To use unlimited jurisdiction policy files by default, place the US_export_policy.jar and the local_policy.jar in the jre/lib/security/ directory of the SDK. Unrestricted SDK JCE policy files are provided for Java 5.0 SR16, Java 6 SR13, Java 6 SR5 (J9 VM2.6), Java 7 SR4, Java 8 GA, and all later releases. Two jar archives are provided by IBM at this location (IBM ID required)
https://www-01.ibm.com/marketing/iwm/iwm/web/reg/pick.do?source=jcesdk&lang=en_US
Download the unrestricted.zip archive and extract the local_policy.jar and US_export_policy.jar files. Place

[27] As of today, IBM Java 8 is available from the Canonical Partner Archive. If IBM Java 7 is required to be used for any purpose, it has to be downloaded from the developerWorks site.

 ©Copyright IBM Corporation 2017

those two files in the <Install_Directory>/jdk/jre/lib/security directory of your Java Runtime environment. Replace the existing files with the same names.

For a default IBM Java Runtime Environment installation on IBM Z, the directory would be:

```
/opt/ibm/java-s390x-80/jre/lib/security
```

6.4.3 Hardware support for encryption in a IBM Java 7 environment

Today IBM Java Version 7 is still widely used in production environments. Therefore we start with a look at this version. Starting with this section we use a slightly modified version of the sample Java test program from [19] (see Example 122) which does AES encryption in ECB mode with a randomly generated key. Because no provider is explicitly specified to be used in this test program, the AES encryption operation will be performed by the provider according the assigned priority in the *java.security* file (compare Example 126 and 129).

```
import java.security.*;
import java.security.spec.*;
import java.security.interfaces.*;
import javax.crypto.*;
import javax.crypto.spec.*;
import javax.crypto.interfaces.*;

class Encrypt0 {

 public static void main (String[] args)
 {
   SecretKey aesKey = null;
   try { // create random AES key
     KeyGenerator keygen = KeyGenerator.getInstance("AES");
     aesKey = keygen.generateKey();
   } catch (Exception e){
               throw(new RuntimeException(e));}
   Cipher aesCipher;
   try { // Create the cipher
     aesCipher =   Cipher.getInstance("AES/ECB/NoPadding");
     // Initialize the cipher for encryption
     aesCipher.init(Cipher.ENCRYPT_MODE, aesKey);
     // Our cleartext
     byte[] cleartext = "0123456789abcdef".getBytes();
     // Print cleartext
        System.out.println("our clear text - to be encrypted:");
     for (int i=0; i<cleartext.length; i++)
     {System.out.print((char) cleartext[i]+", ");};
     System.out.println();
     // Encrypt the cleartext
     byte[] ciphertext = aesCipher.doFinal(cleartext);

    // Print ciphertext
        System.out.println("text encrypted:");
    for (int i=0; i<ciphertext.length; i++)
    {System.out.print("0x" + String.format("%02x", ciphertext[i])+", ");};
    System.out.println();

     // Initialize the same cipher for
     // decryption
     aesCipher.init(Cipher.DECRYPT_MODE, aesKey);
```

 ©Copyright IBM Corporation 2017

```
    // Decrypt the ciphertext
    byte[] cleartext1 = aesCipher.doFinal(ciphertext);
       System.out.println("text decrypted:");
    // Print cleartext1
    for (int i=0; i<cleartext1.length; i++)
    {System.out.print((char) cleartext1[i]+", ");};
    System.out.println();

  } catch (Exception e) {
              throw(new RuntimeException(e));}
  System.out.println("Done!");
 }
}
```

Example 122: Java test program Encrypt0 for AES with ECB mode encryption

In this section as well as in section 6.4.4 and 6.4.5, we run the test program (see Example 122) initially without and then with using the IBMPCKS11Impl provider. For this purpose we simply modify the list of providers in the *java.security* file. Thus, the examples comprise four main runs of our AES encryption sample program with different characteristics described after outlining each run.

At first we initialize the variables *PATH* and *JAVA_HOME* to use the Java 7 environment:

```
export PATH=$PATH:/opt/ibm/java-s390x-71/jre/bin
export JAVA_HOME=/opt/ibm/java-s390x-71/jre
```

Example 123: Setup Java 7 environment

We verify whether the Java 7 environment is setup correctly:

```
ubuntu@zlin42:~$ java -version
java version "1.7.0"
Java(TM) SE Runtime Environment (build pxz6470_27sr3fp60-20161021_01(SR3 FP60))
IBM J9 VM (build 2.7, JRE 1.7.0 Linux s390x-64 Compressed References
   20161005_321280 (JIT enabled, AOT enabled)
J9VM - R27_Java727_SR3_20161005_1253_B321280
JIT  - tr.r13.java_20161003_125478
GC   - R27_Java727_SR3_20161005_1253_B321280_CMPRSS
J9CL - 20161005_321280)
JCL - 20161021_01 based on Oracle jdk7u121-b15
```

Example 124: Verify version of active Java

Now we compile the source (see Example 122) of our test program *Encrypt0.java* using the *javac* command - located in */opt/ibm/java-s390x-71/bin/* - and create the *Encrypt0.class* file for execution as shown in Example 125:

```
ubuntu@zlin42:~$ /opt/ibm/java-s390x-71/bin/javac Enctypt0.java
```

Example 125: Create the class file out of Encrypt0.java

At first we run the test program without any adaptation and modifications to the Java environment. This means we use the default Java security file (i.e. *IBMPKCS11Impl* provider is not available) as shown in Example 126.

```
ubuntu@zlin42:~$ less /opt/ibm/java-s390x-71/jre/lib/security/java.security
...
#
# List of providers and their preference orders (see above):
#
security.provider.1=com.ibm.jsse2.IBMJSSEProvider2
```

 ©Copyright IBM Corporation 2017

```
security.provider.2=com.ibm.crypto.provider.IBMJCE
security.provider.3=com.ibm.security.jgss.IBMJGSSProvider
security.provider.4=com.ibm.security.cert.IBMCertPath
security.provider.5=com.ibm.security.sasl.IBMSASL
security.provider.6=com.ibm.xml.crypto.IBMXMLCryptoProvider
security.provider.7=com.ibm.xml.enc.IBMXMLEncProvider
security.provider.8=com.ibm.security.jgss.mech.spnego.IBMSPNEGO
security.provider.9=sun.security.provider.Sun
. . .
```

Example 126: Providers in default java.security file of Java 7

We reset the libica statistical counters, and the run the test program.

```
ubuntu@zlin42:~$ icastats -r

ubuntu@zlin42:~$ java Encrypt0
our clear text - to be encrypted:
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f,
text encrypted:
0xd1, 0x2d, 0x3c, 0xc4, 0xad, 0x18, 0x2a, 0xa4, 0xca, 0x90, 0xbb, 0x57, 0xe6,
   0x48, 0x20, 0x35,
text decrypted:
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f,
Done!
```

Example 127: Reset counters and run test program in default environment (without IBMPCKS11impl)

After execution of the test program we check the libica statistical counters for a change (see Example 128).

```
ubuntu@zlin42:~$ icastats
 function      |          # hardware        |        # software
-------------+-------------------------+-------------------------
              |       ENC    CRYPT    DEC |        ENC    CRYPT    DEC
-------------+-------------------------+-------------------------
        SHA-1 |                0           |                0
. . .
 DRBG-SHA-512 |                0           |                0
. . .
      AES ECB |          0               0 |          0               0
. . .
```

Example 128: Check for increased counters in default environment (without IBMPCKS11impl)

As expected, the counters for DRBG-SHA-512 and for AES-ECB operations are not incremented (see Example 128) as the libica has not been used.

For completeness, we check whether the *cpacfstats* counters are changed by activities during execution of the test program (similar as we did in section 5.6): As expected, the *cpacfstats* counters are not incremented with our first test in a IBM Java 7 environment. That scheme will be repeated below for runs using the openCryptoki tokens via the IBMPCKS11impl provider.

The second run will demonstrate IBM Java 7 using the IBMPKCS11Impl provider with two different openCryptoki tokens. The token to be used can be configured in the configuration file of the IBMPKCS11Impl provider (in our case: *p11.cfg*, see Example 130 and 133).

There is no need to extend the sample program for entering the User PIN for the ICA Token, since logins are only required when using token specific objects. In our scenario, the program can run without user PIN. The DRBG-SHA-512 as well as the AES ECB counters of libica (displayed by the *icastats* command) are not incremented since the libica is not used in the current setup.

 ©Copyright IBM Corporation 2017

Now we enable the Java environment to use the IBMPKCS11Impl crypto provider which allows to benefit from IBM Z hardware support for encryption by using services via openCryptoki.

For this purpose we modify and adapt the configuration in the *java.security* file to use the IBMPKCS11Impl provider as shown in Example 129.

```
ubuntu@zlin42:~$ less /opt/ibm/java-s390x-71/jre/lib/security/java.security
. . .
#
# List of providers and their preference orders (see above):
#
security.provider.1=com.ibm.crypto.pkcs11impl.provider.IBMPKCS11Impl /root/p11.cfg
security.provider.2=com.ibm.jsse2.IBMJSSEProvider2
security.provider.3=com.ibm.crypto.provider.IBMJCE
security.provider.4=com.ibm.security.jgss.IBMJGSSProvider
security.provider.5=com.ibm.security.cert.IBMCertPath
security.provider.6=com.ibm.security.sasl.IBMSASL
security.provider.7=com.ibm.xml.crypto.IBMXMLCryptoProvider
security.provider.8=com.ibm.xml.enc.IBMXMLEncProvider
security.provider.9=com.ibm.security.jgss.mech.spnego.IBMSPNEGO
security.provider.10=sun.security.provider.Sun
. . .
```

Example 129: IBMPKCS11Impl provider at first position in java.security of Java 7

The IBMPKCS11Impl provider should be listed as the first provider in the list. This ensures that the IBMPKCS11Impl provider is the first one that is asked to service a specific cryptographic request. The IBMPKCS11Impl provider has a configuration profile to determine the scope of the provider. In our case name the configuration file *p11.cfg* and store it in the directory */root/*. Note: The full path/filename of the configuration file must be separated by one space character after IBMPKCS11Impl provider declaration in one single line of the *java.security* file. Note: There must be no carriage return/line feed between the provider name IBMPKCS11Impl and the token configuration path and file name! Otherwise the configuration file is not recognized.

The content of the *p11.cfg* file determines the openCryptoki token to be used. In our tests, we use the Soft Token *myswtoken* (defined in section 6.1.2) and the ICA Token *myicatoken* (defined in section 6.1.1).

For a first verification with the IBMPKCS11Impl provider we use the software token (*myswtoken*) of our openCryptoki environment (see Example 130).

```
name = Sample
description = Sample IBMPKCS11Impl config file for Linux on z Systems
library = /usr/lib/pkcs11/PKCS11_API.so
# reference to SoftToken Slot via tokenlabel
tokenlabel = mysofttoken
```

Example 130: Configuration of IBMPKCS11Impl provider - use software token

The line configuring the tokenlabel must conform to the label in the openCryptoki token (compare with Example 131). The label is set during token initialization as shown in Example 101.

```
$ pkcsconf -t -c 3
Token #3 Info:
        Label: myswtoken
. . .
```

Example 131: PKCS#11 configuration - slot 3 contains software token

Execution of the test program results in the same way as above, we get the same behaviour. The counters for AES-ECB are not incremented as the libica has not been used (see Example 132).

 ©Copyright IBM Corporation 2017

```
ubuntu@zlin42:~$ icastats -r

ubuntu@zlin42:~$ java Encrypt0
our clear text - to be encrypted:
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f,
text encrypted:
0xc3, 0x5a, 0x62, 0xfb, 0x33, 0xf9, 0xd7, 0x03, 0x3f, 0x66, 0xf8, 0x15, 0xdb,
   0xfd, 0x55, 0x3b,
text decrypted:
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f,
Done!

ubuntu@zlin42:~$ icastats
 function      |          # hardware       |        # software
---------------+---------------------------+-------------------------
               |       ENC    CRYPT   DEC |        ENC    CRYPT   DEC
---------------+---------------------------+-------------------------
        SHA-1  |                0          |                0
. . .
 DRBG-SHA-512  |              169          |                0
. . .
      AES ECB  |         0              0  |         0              0
. . .
```

Example 132: Reset counters - run test using IBMPCKS11impl provider with software token and verify counters

Next we configure the use of the openCryptoki ICA token (*myicatoken*) by modifying the p11.cnf file (see Example 133). The software stack used by the IBMPCKS11Impl provider is extended by the *libica* library, which uses AES cryptographic functions accelerated by CPACF.

```
name = Sample
description = Sample IBMPKCS11Impl config file for Linux on z Systems
library = /usr/lib/pkcs11/PKCS11_API.so
# reference to ICA Slot via tokenlabel
tokenlabel = myicatoken
```

Example 133: Configuration of IBMPKCS11Impl provider - use ICA token

The line configuring the tokenlabel must conform to the label in the openCryptoki token (compare with Example 134).

```
ubuntu@zlin42:~$ pkcsconf -t -c 1
Token #1 Info:
        Label: myicatoken
. . .
```

Example 134: PKCS#11 configuration - slot 1 contains ICA token

Running the test program in the same way as above, we can prove that CPACF support has been used via libica. The counters for AES-ECB are incremented as the libica has been used (see Example 135).

```
ubuntu@zlin42:~$ icastats -r

ubuntu@zlin42:~$ java Encrypt0
. . .
0xba, 0xec, 0x9c, 0x28, 0x0a, 0x0f, 0x9c, 0xa8, 0x21, 0x92, 0x32, 0x69, 0x41,
   0x40, 0x50, 0xc9,
```

 ©Copyright IBM Corporation 2017

```
. . .

ubuntu@zlin42:~$ icastats
 function      |          # hardware        |        # software
-------------+-------------------------+-------------------------
              |       ENC    CRYPT    DEC |        ENC    CRYPT    DEC
-------------+-------------------------+-------------------------
        SHA-1 |                0           |                0
. . .
 DRBG-SHA-512 |              189           |                0
. . .
      AES ECB |          1               1 |          0               0
. . .
```

Example 135: Reset counters - run test using IBMPCKS11Impl provider with ICA token and verify counters

In this section we could show that we benefit from hardware support of IBM Z for encryption workload. We achieved this by positioning the IBMPKCS11Impl provider in the first position of the search order for providers and using the ICA token for openCryptoki.

6.4.4 Hardware support for encryption in a IBM Java 8 environment

In IBM Java 8 some significant enhancements have been implemented. Besides a lot of general performance enhancements, IBM Java 8 SR1 offers significant improvements to IBMJCE. The IBMJCE provider now automatically detects and exploits an on-core hardware cryptographic accelerator (CPACF) as well as the Single Instruction, Multiple Data (SIMD) vector engine available with IBM z13 or later. These enhancements allow the default IBMJCE provider to automatically use CPACF support for AES, DES, Tripple DES and SHA algorithms without the need to use the special IBMPKCS11Impl provider for hardware acceleration. However, it still makes sense to configure the IBMPKCS11Impl provider to also use hardware acceleration for other algorithms like RSA.

In this section, we will repeat the same tests as already done in the Java 7 environment as described in section 6.4.3 with the providers as specified in the default java.security file and with using the IBMPKCS11Impl provider. Below in section 6.4.5 we will modify the configuration of the IBMPKCS11Impl provider to exclude some algorithms from execution with IBMPKCS11Impl and use the CPACF support of the default provider IBMJCE.

The list of providers in the default *java.security* file of IBM Java 8 (see Example 136) is identical with IBM Java 7.

```
ubuntu@zlin42:~$ less /opt/ibm/java-s390x-80/jre/lib/security/java.security
. . .
#
# List of providers and their preference orders (see above):
#
security.provider.1=com.ibm.jsse2.IBMJSSEProvider2
security.provider.2=com.ibm.crypto.provider.IBMJCE
. . .
security.provider.9=sun.security.provider.Sun
. . .
```

Example 136: Providers in default java.security file of Java 8

For the test with our test program *Encrypt0* we we initialize the variables *PATH* and *JAVA_HOME* to use the Java 8 environment

```
export PATH=$PATH:/opt/ibm/java-s390x-80/jre/bin
export JAVA_HOME=/opt/ibm/java-s390x-80/jre
```

Example 137: Setup Java 8 environment

 ©Copyright IBM Corporation 2017

and verify whether the Java 8 environment is setup correctly:

```
ubuntu@zlin42:~$ java -version
java version "1.8.0"
Java(TM) SE Runtime Environment (build pxz6480sr3fp22-20161213_02(SR3 FP22))
IBM J9 VM (build 2.8, JRE 1.8.0 Linux s390x-64 Compressed References
   20161209_329148 (JIT enabled, AOT enabled)
J9VM - R28_20161209_1345_B329148
JIT  - tr.r14.java.green_20161207_128946
GC   - R28_20161209_1345_B329148_CMPRSS
J9CL - 20161209_329148)
JCL - 20161213_01 based on Oracle jdk8u111-b14
```

Example 138: Verify version of active Java

Now we reset the *icastats* counters and run the test program in the default environment (i.e. without IBMPCKS11impl)

```
ubuntu@zlin42:~$ icastats -r

ubuntu@zlin42:~$ java Encrypt0
our clear text - to be encrypted:
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f,
text encrypted:
0xa0, 0x5a, 0x41, 0xb3, 0x37, 0x7f, 0x1f, 0x43, 0xfd, 0x6f, 0xc9, 0x59, 0xda,
   0x17, 0x6f, 0x33,
text decrypted:
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f,
Done!

ubuntu@zlin42:~$ icastats
 function      |          # hardware       |        # software
-------------+-------------------------+-------------------------
              |       ENC    CRYPT   DEC |        ENC    CRYPT   DEC
-------------+-------------------------+-------------------------
        SHA-1 |                0          |                0
. . .
 DRBG-SHA-512 |                0          |                0
. . .
      AES ECB |          0              0 |          0              0
. . .
```

Example 139: Reset counters - run test program in default environment (without IBMPCKS11impl) and verify counters

As expected, all the *icastats* counters remain 0 as no requests has been passed via the IBMPKCS11Impl provider to the PKCS#11 interface using the library *libica*.

Now we check also whether the *cpacfstats* counters are changed by activities during execution of the test program (similar as we did in section 5.6): As expected, the *cpacfstats* counters for AES and SHA are incremented as the default provider IBMJCE has built-in functionality to use CPACF. Now we enable the java environment to use the IBMPKCS11Impl provider which uses the PKCS#11 interface (openCryptoki) to execute the cryptographic operation requests. For this purpose we modify and adapt the configuration in the *java.security* file to use the IBMPKCS11Impl provider

```
ubuntu@zlin42:~$ less /opt/ibm/java-s390x-80/jre/lib/security/java.security
. . .
#
# List of providers and their preference orders (see above):
#
```

 ©Copyright IBM Corporation 2017

```
security.provider.1=com.ibm.crypto.pkcs11impl.provider.IBMPKCS11Impl /root/p11.cfg
security.provider.2=com.ibm.jsse2.IBMJSSEProvider2
security.provider.3=com.ibm.crypto.provider.IBMJCE
. . .
security.provider.10=sun.security.provider.Sun
. . .
```

Example 140: IBMPKCS11Impl provider at first position in java.security of Java 8

For the first run in this environment with the IBMPKCS11Impl provider, we use the software token (*myswtoken*) of our openCryptoki environment as shown in Example 130.

We reset the *icastats* counters, run the test program and check the counters again:

```
ubuntu@zlin42:~$ icastats -r

ubuntu@zlin42:~$ java Encrypt0
our clear text - to be encrypted:
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f,
text encrypted:
0xda, 0x86, 0xf3, 0x0d, 0xab, 0x71, 0x00, 0xe3, 0x8a, 0x90, 0x4c, 0xd5, 0x94,
   0xde, 0x30, 0x41,
text decrypted:
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f,
Done!

ubuntu@zlin42:~$ icastats
 function      |          # hardware       |        # software
-------------+-------------------------+-------------------------
              |       ENC    CRYPT   DEC |        ENC    CRYPT   DEC
-------------+-------------------------+-------------------------
        SHA-1 |                0          |                0
. . .
 DRBG-SHA-512 |              169          |                0
. . .
      AES ECB |         0               0 |         0              0
. . .
```

Example 141: Reset counters - run test using IBMPCKS11impl provider with software token and verify counters

As expected, the *icastats* counters for AES ECB remains 0, as the AES request has not been passed to *libica* as the software token has been used.

Finally, we want to use the ICA token to pass the AES encryption request via libica to the CPACF. In the configuration file of the IBMPKCS11Impl provider we specify to use the ICA token (*myicatoken*) as shown in Example 133.

The check for the *icastats* counters after execution of the test program

```
ubuntu@zlin42:~$ icastats -r

ubuntu@zlin42:~$ java Encrypt0
our clear text - to be encrypted:
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f,
text encrypted:
0xac, 0xfc, 0x3e, 0xd9, 0xd7, 0x2d, 0xed, 0xec, 0x77, 0xbb, 0xa8, 0x64, 0xdc,
   0x5c, 0xc8, 0xc9,
text decrypted:
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f,
Done!
```

 ©Copyright IBM Corporation 2017

```
ubuntu@zlin42:~$ icastats
 function      |          # hardware        |        # software
-------------+-------------------------+-------------------------
              |       ENC    CRYPT    DEC |        ENC    CRYPT    DEC
-------------+-------------------------+-------------------------
        SHA-1 |                0          |                0
. . .
 DRBG-SHA-512 |              188          |                0
. . .
      AES ECB |          1              1 |          0              0
. . .
```

Example 142: Reset counters - run test using IBMPCKS11impl provider with ICA token and verify counters

shows that now the AES ECB counters are now increased, as the ICA token passes the encryption request via libica to the CPACF.

For completeness we check also whether the *cpacfstats* counters are changed by activities during execution of the test program (similar as we did in section 5.6): As expected, the *cpacfstats* counters for AES and SHA are incremented.

6.4.5 IBM Java 8: Using hardware acceleration for AES and RSA with two providers

As IBM Java 8 provides built-in capability to use CPACF support for AES algorithms, it is a possible to use this support from the default IBMJCE provider and to benefit from hardware support for encryption with RSA by using the IBMPKCS11Impl provider. This might result in a slight performance difference. To use the IBMPKCS11Impl provider for SHA,DES, Tripple DES and RSA only, the AES has to be disabled in the provider profile. In our case we disable *CKM_AES_ECB* and *CKM_AES_KEY_GEN* and all other supported AES algorithms (see [22]) from the configuration of the IBMPKCS11Impl provider as shown in Example 143.

```
name = Sample
description = Sample IBMPKCS11Impl config file for Linux on z Systems
library = /usr/lib/pkcs11/PKCS11_API.so
# reference to ICA Slot via tokenlabel
tokenlabel = myicatoken
disabledmechanisms = {
 CKM_AES_ECB
 CKM_AES_KEY_GEN
 CKM_AES_CBC
 CKM_AES_CBC_PAD
}
```

Example 143: Configuration of IBMPKCS11Impl provider for Java 8 - exclude AES when using ICA token

With the configuration of the IBMPKCS11Impl provider from example 143 in a IBM Java 8 environment, hardware support for AES encryption is now provided by the default IBMJCE provider from IBM Java 8. For the other algorithms the hardware support of CPACF and Crypto Express feature the IBMPKCS11Impl provider is used. With this configuration the AES counters of *icastats* are not increased (AES is not executed by IBMPKCS11Impl provider via openCryptoki and libica, but with default provider). If we check with *cpacfstats* command we can see, that the AES requests of our test program are executed via CPACF support.

As IBMJCE also includes CPACF acceleration for SHA, DES and Tripple DES, it is also an option to consider to exclude also these algorithms from execution by IBMPKCS11Impl provider.

 ©Copyright IBM Corporation 2017

7 Conclusion

Using modern Linux distributions on IBM Z or LinuxONE, a relatively low effort is required in order to enable a Linux Server to use existing hardware capabilities to accelerate encryption operations for OpenSSH via OpenSSL. This advantage applies not only to SSH sessions, but also to Secure Copy (SCP), SFTP and rsync. The easiest way to benefit from this advantage is to adapt SSH and SSHD profiles and place those algorithms supported by CPACF at the top of the search order, or - if possible - exclude algorithms which do not use hardware support from CPACF or Crypto Express features. Especially a good configuration of the client has the most effect.

Once a Linux server is enabled to use CPACF including or not Crypto Express feature support, there is also very little configuration effort required for Apache web server to use hardware cryptographic support for secure data transfer. Applications and programs that use PKCS#11 interface for encryption can also use hardware accelerated encryption support to speed up and save a lot of cycles. This helps also in IBM Java environments where only minimal configuration effort is required.

The security architecture of Java is versatile and can be adapted to security requirements. To reduce the overhead, hardware acceleration for cryptographic functions is at hand. To achieve this, some modifications have to be done to use the required providers. With IBM Java7 per default only a software encryption is available. However, in a IBM Java 8 environment nothing has to be configured to benefit from acceleration and high speed encryption and decryption support for clear key operations - at least for SHA, AES, DES and Tripple DES algorithms. Considering Figure 2, it becomes evident that the built-in support has a shorter pathlength than using PKCS#11 interface, providing potential performance benefits.

 ©Copyright IBM Corporation 2017

Source code of java program sample

```
import java.security.*;
import java.security.spec.*;
import java.security.interfaces.*;
import javax.crypto.*;
import javax.crypto.spec.*;
import javax.crypto.interfaces.*;

class Encrypt0 {
 public static void main (String[] args)
 {
   SecretKey aesKey = null;
   try { // create random AES key
     KeyGenerator keygen = KeyGenerator.getInstance("AES");
     aesKey = keygen.generateKey();
   } catch (Exception e){
               throw(new RuntimeException(e));}
   Cipher aesCipher;
   try { // Create the cipher
     aesCipher =   Cipher.getInstance("AES/ECB/NoPadding");
     // Initialize the cipher for encryption
     aesCipher.init(Cipher.ENCRYPT_MODE, aesKey);
     // Our cleartext
     byte[] cleartext = "0123456789abcdef".getBytes();
     // Print cleartext
        System.out.println("our clear text - to be encrypted:");
     for (int i=0; i<cleartext.length; i++)
     {System.out.print((char) cleartext[i]+", ");};
     System.out.println();
     // Encrypt the cleartext
     byte[] ciphertext = aesCipher.doFinal(cleartext);

    // Print ciphertext
        System.out.println("text encrypted:");
    for (int i=0; i<ciphertext.length; i++)
    {System.out.print("0x" + String.format("%02x", ciphertext[i])+", ");};
    System.out.println();

     // Initialize the same cipher for
     // decryption
     aesCipher.init(Cipher.DECRYPT_MODE, aesKey);

     // Decrypt the ciphertext
     byte[] cleartext1 = aesCipher.doFinal(ciphertext);
        System.out.println("text decrypted:");
     // Print cleartext1
     for (int i=0; i<cleartext1.length; i++)
     {System.out.print((char) cleartext1[i]+", ");};
     System.out.println();

   } catch (Exception e) {
               throw(new RuntimeException(e));}
   System.out.println("Done!");
 }
}
```

 ©Copyright IBM Corporation 2017

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.

Klaus Bergmann is a Senior IT Specialist in the IBM Client Center in the IBM Boeblingen Lab, Germany. He holds a master degree in Computer Science from the University of Bonn, Germany. He has extensive knowledge of IBM Z hardware, virtualization, and Linux. He also works on Blockchain technology.

Reinhard Buendgen is an architect for cryptography and RAS in the Linux on z Systems development team in the IBM Boeblingen Lab, 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 IBM Z. (IBM z/OS, IBM z/VM, IBM z/VSE®, Linux on z Systems, and various subsystems). He also prepares demonstrations on IBM Z.

Jonathan Furminger is a Software Engineer working at IBM Poughkeepsie. Currently he works on Java Security for z/OS and Linux for z Systems. Previously he worked on z/OS LDAP. Jonathan holds a Bachelor and Masters Degree from Binghamton University. He has been working for IBM for 27 years.

Frank Heimes is working in the IT industry since several decades. He holds a degree on Computer Science (minor in electrical and electronic engineering) and worked in industry automation, IT conulting and IT development. He started at Siemens AG, worked afterwards for 15 years at IBM Research and Development Germany as Senior IT Architect (Master IT Architect according to TOG) and Team lead of the IBM Z Advanced Technical Skills (Z ATS) team, now being the Technical Lead for Ubuntu on s390x development at Canonical Ltd.

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 15 years. Currently he is involved in several Linux for z Systems Proof-of-Concept projects and LinuxONE customer projects running at the IBM Client Center.

Christian Rund is an IT Specialist with test focus at the IBM Lab in Boeblingen, Germany. He holds a degree in Computer Science (Diplom) from University of Stuttgart, Germany. He joined Linux for z Systems Distribution Test two years ago. Before, he contributed as developer and tester to Hardware and Firmware development projects.

Patrick Steuer is a crypto programmer for Linux on z Systems at the IBM Lab in Boeblingen, Germany. He holds a degree in mathematics (Diplom) from the University of Kaiserslautern, Germany. Currently he is working on the vectorization of cryptographic algorithms to exploit the z/Architecture's vector facility.

Arwed Tschoeke is a member of Z ATS 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.

Acknowledgement

Our very best acknowledgement for discussions and helpful hints belongs to
Boris Barth, Marc Beyerle, Harald Freudenberger, Thomas Hanicke, Stephan Hartig, Gerald (Jerry) Hosch, Elisabeth Puritscher, Joerg Schmidbauer, Ingo Tuchscherer, Klaus Werner, Arthur Winterling, as well as to
Agnès Gnirss, Justyna Steuer and Heyke Diddens-Tschoeke.

Acronyms

3DES Triple DES

 ©Copyright IBM Corporation 2017

ADH/AECDH Anonymous (Elliptic Curve) Diffie-Hellman

AEAD Authenticated Encryption with Associated Data

AES Advanced Encryption Standard

AP Adjunct Processor

API Application Programming Interface

BSI Bundesamt für Sicherheit in der Informationstechnik (Federal Office for Information Security)

CBC Cipher Block Chaining

CCA Common Cryptographic Architecture

CCM Counter with CBC-MAC

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

CP Central Processor

CPACF Central Processor Assist for Cryptographic Functions

CPU Central Processing Unit

CTR Counter Mode

DES Data Encryption Standard

DH Diffie–Hellman

DSA Digital Signature Algorithm

ECDHE/DHE Ephemeral (Elliptic Curve) Diffie-Hellman

EP11 IBM Enterprise PKCS#11

ECB Electronic Codebook

FQDN Fully Qualified Domain Name

FTP File Transfer Protocol

GCM Galois/Counter Mode

HMAC Hash-based Message Authentication Code

HMC Hardware Management Console

HTTP Hypertext Transfer Protocol

HTTPS Hypertext Transfer Protocol Secure

 ©Copyright IBM Corporation 2017

ICC IBM Crypto for C

ICA IBM Crypto API

IPL Initial Program Load

ISP Internet Service Provider

JCA/JCE Java Cryptography Architecture / Java Cryptography Extension

JDK Java Development Kit

JRE Java Runtime Environment

LIC Licensed Internal Code

LPAR Logical partition

LTS Long Term Support

MAC Message Authentication Code

MD5 Message-Digest algorithm 5

NIST National Institute of Standards and Technology

OFB Output Feedback

PCBC propagating Cipher Block Chaining

PFS Perfect Forward Security

PIN Personal Identification Number

PKCS Public Key Cryptography Standard

PRNG Pseudo Random Number Generator

RHEL Red Hat Enterprise Linux

RSA Rivest, Shamir and Adleman algorithm

SCP Secure Copy Protocol

SDK Software Developers Kit

SE Support Element

SELinux Security-Enhanced Linux

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

SHA Secure Hash Algorithm

SIMD Single Instruction, Multiple Data

SLES SUSE Linux Enterprise Server

SO Serurity Officer

SP Service Pack

SSH Secure Shell

 ©Copyright IBM Corporation 2017

SSHD Secure Shell Daemon

SSL Secure Sockets Layer

TDES Triple DES

TLS Transport Layer Security

TPM Trusted Platform Module

TSS Trusted Computing Group's Software Stack

References

[1] 8 challenges that keep financial services CTOs and CIOs up at night, by Jennifer Lonoff Schiff
www.cio.com/article/3128314/financial-it/8-challenges-that-keep-financial-services-ctos-and-cios-up-at-night.html

[2] 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

[3] Hardware cryptographic support of IBM z Systems for OpenSSH in RHEL 7.2 and SLES 12 SP1, by Uwe Denneler, Harald Freudenberger, Paul Gallagher, Manfred Gnirss, Guillaume Hoareau, Arwed Tschoeke, Ingo Tuchscherer, Arthur Winterling.
https://www-03.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP102653

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

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

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

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

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

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

[10] z/VM V6.4 CP Planning and Administration, SC24-6178-10

[11] 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

[12] Ubuntu version 16.04 point 1 is out
https://insights.ubuntu.com/2016/07/28/ubuntu-version-16-04-point-1-is-out/

[13] Linux on z Systems and LinuxONE: Device Drivers, Features, and Commands on Ubuntu Server 16.04 LTS
http://public.dhe.ibm.com/software/dw/linux390/docu/lux1dd00.pdf

 ©Copyright IBM Corporation 2017

[14] IBM Knowledge Center: cpacfstats - Monitor CPACF cryptographic activity
https://www.ibm.com/support/knowledgecenter/linuxonibm/com.ibm.linux.z.ludd/ludd_r_cpacfstats_cmd.html

[15] User Manual - mod_ssl version 2.8, chapter 3: Reference
http://www.modssl.org/docs/2.8/ssl_reference.html

[16] NIST's Policy on Hash Functions, August 5, 2015
http://csrc.nist.gov/groups/ST/hash/policy.html

[17] Using Linux on System z Hardware Cryptography With the PKCS#11 Cryoptography Stack, by Reinhard Buendgen in Enterprise Tech Journal on October 6, 2014
http://enterprisesystemsmedia.com/article/using-linux-on-system-z-hardware-cryptography-with-the-pkcs11-cryoptography#sr=g&m=o&cp=or&ct=-tmc&st=(opu%20qspwjefe)&ts=1485100746

[18] Configuring an Apache mod_nss server to exploit z Systems cryptographic hardware, by Patrick Steuer, Reinhard Buendgen, George C. Wilson
http://www.ibm.com/support/knowledgecenter/linuxonibm/liaag/wnsf/l0wnsf00_2015.htm

[19] Using Crypto Hardware With Java in Linux on System z, by Reinhard Buendgen, Peter Spera in Enterprise Tech Journal on March 20, 2013
http://enterprisesystemsmedia.com/article/using-crypto-hardware-with-java-in-linux-on-system-z#sr=g&m=o&cp=or&ct=-tmc&st=(opu%20qspwjefe)&ts=1485100746

[20] Having a cup of Java on Ubuntu for IBM z Systems and LinuxONE
http://ubuntu-on-big-iron.blogspot.com/2017/04/having-cup-of-java-on-ubuntu-for-ibm-z.html

[21] Security Guide - IBM SDK Policy files
https://www.ibm.com/support/knowledgecenter/en/SSYKE2_8.0.0/com.ibm.java.security.component.80.doc/security-component/sdkpolicyfiles.html

[22] Supported cryptographic hardware for the IBMPKCS11Impl provider in IBM SDK for Linux on z Systems architecture, Java Technology Edition, Version 8
http://www-01.ibm.com/support/docview.wss?uid=swg21967855

Trademarks

The following terms are trademarks of the International Business Machines Corporation in the United States, or other countries, or both: developerWorks®, Express®, IBM®, IBM LinuxONE™, IBM LinuxONE Emperor™, IBM LinuxONE Rockhopper™, IBM Z®, IBM z13®, IBM z13s™, IBM z Systems®, System z®, System z10®, WebSphere®, z10™, z13™, z13s™, z Systems®, z/OS®, z/VM®, z/VSE®.
Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.
Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.
SUSE and SLES are registered trademarks of Novell, Inc. in the United States and other countries.
Red Hat, the Shadowman logo, Red Hat Enterprise Linux, RHEL, Red Hat Network, and RHN are trademarks or registered trademarks of Red Hat, Inc. in the United States and other countries.
Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.
Apache is a trademark of Apache Software Foundation (ASF)
Canonical and Ubuntu are a registered trademarks of Canonical Ltd.
Other company, product, or service names may be trademarks or service marks of others.

 ©Copyright IBM Corporation 2017