

ISC

BIND 9

9.18.14

Integration Guide

CryptoServer HSM

Imprint

Copyright 2026	Utimaco IS GmbH Krefelder Straße 220 52070 Aachen Germany
Phone	AMERICAS +1-844-UTIMACO (+1 844-884-6226) EMEA +49 800-627-3081 APAC +81 800-919-1301
Internet	https://support.hsm.utimaco.com/
e-mail	support@utimaco.com
Document Version	1.1.0
Date	2026-08-12
Status	PUBLISHED
Document No.	IG-2026-0004
All rights reserved	No part of this documentation may be reproduced in any form (printing, photocopy or according to any other process) without the written approval of Utimaco IS GmbH or be processed, reproduced or distributed using electronic systems. Utimaco IS GmbH reserves the right to modify or amend the documentation at any time without prior notice. Utimaco IS GmbH assumes no liability for typographical errors and damages incurred due to them. All trademarks and registered trademarks are the property of their respective owners.

Table of Contents

1	Introduction	5
1.1	About This Guide	5
1.2	Target Audience for This Guide	5
1.3	Document Conventions	5
1.4	Abbreviations	6
2	Overview	8
2.1	Bind9	8
2.2	Utimaco CryptoServer HSM.....	8
2.3	Utimaco u.trust Anchor	8
3	Integration Requirements and Prerequisites	9
3.1	Tested Versions.....	9
3.2	Software Requirements.....	9
3.3	Hardware Requirements.....	10
3.4	Prerequisites	10
4	Installing and Configuring Utimaco SecurityServer Software	12
4.1	Download and Install Utimaco Software	12
4.2	SecurityServer PKCS#11 Configuration	13
4.3	Create SO User and Initialize a Slot.....	14
5	Configuring OpenSSL to Use Utimaco SecurityServer	16
5.1	Installing Libp11	16
5.2	Setting up Utimaco SecurityServer Library in OpenSSL Configuration File.....	17
5.2.1	Verifying PKCS#11 Engine	18
6	Installing and Running Bind9.....	19
7	Integrating Bind9 with CryptoServer HSM	28
7.1	Generate ZSK and KSK on CryptoServer HSM.....	28
7.1.1	Generating KSK and ZSK with RSA Algorithm.....	28
7.1.2	Generating KSK and ZSK with ECC Algorithm	30
7.2	Generate Reference Key files of HSM Keys for Bind9	32
7.2.1	For RSA Keys.....	32
7.2.2	For ECC Keys	34
7.3	Sign and Verify the Zone File.....	36

7.3.1	Sign and Verify Zone Files with RSA Keys	36
7.3.2	Sign and Verify Zone Files with ECC Keys	37
7.4	Configuring Bind9 to Use Signed Zone File	38
7.5	Key Rollover for KSK and ZSK	39
7.5.1	Generate New KSK and ZSK	39
7.5.2	KSK Rollover	40
7.5.3	ZSK Rollover	42
8	Troubleshooting	46
9	Further Information	47
10	References	48
11	Contact and Support Information	49

1 Introduction

This guide is part of the information and support provided by Utimaco. Additional documentation produced to support your CryptoServer HSM product can be found in the document directory of the CryptoServer HSM product bundle.

All Utimaco SecurityServer product documentation is available from Utimaco's website at <https://utimaco.com/>.

1.1 About This Guide

This guide provides an integration guide explaining how to integrate CryptoServer HSM with Bind9. CryptoServer HSM securely generates and stores the KSK and ZSK keys required by bind9 to sign the zone files.

1.2 Target Audience for This Guide

This guide is intended for administrators of Bind9 and of Utimaco HSMs.

1.3 Document Conventions

The following conventions are used in this guide:

Convention	Use	Example
Bold	Items of the Graphical User Interface (GUI), e.g., menu options	Select Details and click on Properties button
<code>Monospaced</code>	Code that is given for explanation or as an example, file paths	<code>certreq.exe -new request.inf IISCertRequest.csr</code>
<i>Italic</i>	References and important terms	Operating system listed in <i>Tested Versions</i>

Table 1: Document conventions

We use special icons to highlight the most important notes and information.



Here you will find important safety information that should be followed.



Here you will find additional notes or supplementary information.



This message marks the result expected after the successful execution of an instruction.

1.4 Abbreviations

The following abbreviations are used in this guide:

Abbreviation	Meaning
BIND	Berkeley Internet Name Domain
CSADM	CryptoServer Command-line Administration Tool
DNS	Domain Name System
DNSSEC	Domain Name System Security Extensions
GUI	Graphical User Interface
HSM	Hardware Security Module
IP	Internet Protocol

Abbreviation	Meaning
KSK	Key-signing Key
LAN	Local Area Network
PCI-e	PCI Express Interface
PIN	Personal Identification number
PKCS#11	Public-Key Cryptography Standard #11
RSA	Rivest-Shamir-Adleman
SO	Security Officer
URL	Uniform Resource Locator
ZSK	Zone-signing Key
URL	Uniform Resource Locator
TTL	Time to Live

Table 2: List of abbreviations

2 Overview

2.1 Bind9

BIND9 is an open-source DNS server software used for secure DNS services. It supports DNSSEC, dynamic updates, zone transfers, and load. The Berkeley Internet Name Domain (BIND) software implements a domain name server for several operating systems. BIND 9 can be configured as an authoritative name server, a resolver and, on supported hosts, a stub resolver. While large operators usually dedicate DNS servers to a single function per system, smaller operators will find that BIND 9's flexible configuration features support multiple functions, such as a single DNS server acting as both an authoritative name server and a resolver.

2.2 Utimaco CryptoServer HSM

CryptoServer HSM is a hardware security module developed by Utimaco IS GmbH. CryptoServer HSM is a physically protected specialized computer unit designed to perform sensitive cryptographic tasks and to securely manage, as well as store, cryptographic keys and data. It can be used as a universal, independent security component for heterogeneous computer systems.

2.3 Utimaco u.trust Anchor

u.trust Anchor is the next generation hardware security module platform developed by Utimaco IS GmbH. u.trust Anchor is a physically protected specialized computer unit designed for true multi-tenancy, and performing sensitive cryptographic tasks, and to securely manage, as well as store, cryptographic keys and data. It can be used as a universal, independent security component for heterogeneous computer systems.

3 Integration Requirements and Prerequisites

Ensure the system environment you will be using meets the following hardware and software requirements.

3.1 Tested Versions

The integrations that have been successfully tested with the Utimaco HSM and Bind9.

Operating System	Bind9 Version	Utimaco Security Server Version	Utimaco HSM
RHEL 8	9.18.14	SecurityServer 4.51.0.1	CryptoServer CSe-Series/Se-Series
RHEL9	9.18.14	Security server 4.51.0.1	CryptoServer CSe-Series/Se-Series u.trust Anchor Se*k and u.trust Anchor CSAR

Table 3: List of tested versions

3.2 Software Requirements

Software	Software Requirements
OpenSSL	3.0.7 and 1.1.1k
Libp11	0.4.12
HSM Utility	PKCS#11 Tool Version 2 (p11tool2)

Software	Software Requirements
HSM Interfaces	SecurityServer PKCS#11 Provider

Table 4: List of software requirements

3.3 Hardware Requirements

Hardware	Hardware Requirements
Utimaco LAN HSM	CryptoServer CSe-Series/Se-Series LAN with firmware SecurityServer 4. 51.0.1 or higher u.trust Anchor Se*k and u.trust Anchor CSAR with firmware 4.51.0.1 or higher
Utimaco PCI-e HSM	CryptoServer CSe-Series/Se-Series PCI-e with firmware SecurityServer 4. 51.0.1 or higher u.trust Anchor Se*k and u.trust Anchor CSAR with firmware 4.51.0.1 or higher

Table 5: List of hardware requirements



Set up an account on the Utimaco support portal and request download access at the following URL: <https://support.hsm.utimaco.com/>.

3.4 Prerequisites

Before you begin, please ensure that:

- The operating system is listed in *Tested Versions*.
- The CryptoServer HSM is listed in *Tested Versions*.
- The HSM Default Admin is replaced with a new admin user.

- The CryptoServer HSM is set up and configured. Refer to the CryptoServer HSM documentation to set up the HSM.
- The admin access is set up as it is required to install the software.
- You familiarize yourself with the Bind9 documents and setup process.
- You allow port 53 through firewall.

4 Installing and Configuring Utimaco SecurityServer Software

This chapter includes the following sections:

- [Download and Install Utimaco Software](#)
- [SecurityServer PKCS#11 Configuration](#)
- [Create SO User and Initialize a Slot](#)

4.1 Download and Install Utimaco Software

If you have not already done so, please create and request an Utimaco Support Portal Account. This will allow you to download the software components needed for this installation.

1. Copy the downloaded software at the appropriate location on the Bind9 Server.
2. Create utimaco folder under /opt directory and further create 2 directories `/opt/utimaco/bin` and `/opt/utimaco/lib`.

>_ Console

```
# mkdir -p /opt/utimaco/bin  
# mkdir -p /opt/utimaco/lib
```

3. Copy pkcs11 library file `libcs_pkcs11_R3.so` from Utimaco SecurityServer software to the `/opt/utimaco/lib` directory.

>_ Console

```
# cp ~/path_to_application_folder/lib/libcs_pkcs11_R3.so /opt/utimaco/lib
```

4. Copy the csadm and p11tool2 files from Utimaco SecurityServer software to `/opt/utimaco/bin` directory and make both the files executable.

>_ Console

```
# cd ~/path_to_application_folder  
  
# cp csadm p11tool2 /opt/utimaco/bin  
  
# chmod +x /opt/utimaco/bin/csadm /opt/utimaco/bin/p11tool2
```

4.2 SecurityServer PKCS#11 Configuration

1. Create the directory `/etc/utimaco`. Locate the Utimaco PKCS#11 configuration file in your SecurityServer directory, `Linux/x86-64/Crypto_APIS/PKCS11_R3/sample`. Copy the Utimaco PKCS#11 configuration file `cs_pkcs11_R3.cfg` into `/etc/utimaco` directory.

>_ Console

```
# mkdir /etc/utimaco  
  
# cd <install directory>/Software/Linux/x86-64/Crypto_APIs/PKCS11_R3/sample  
  
# cp cs_pkcs11_R3.cfg /etc/utimaco  
  
# cd /etc/utimaco
```

2. Edit the `cs_pkcs11_R3.cfg` file and make the appropriate changes to the file.

cs_pkcs11_R3.cfg

```
[Global]
# For unix:
Logpath = /tmp
# Loglevel (0 = NONE; 1 = ERROR; 2 = WARNING; 3 = INFO; 4 = TRACE)
Logging = 1
Keepalive = false
# Set the Device to connect with
[CryptoServer]
# Device specifier
Device = <HSM_IP>
```



For more information regarding the commands and command parameters please check the Utimaco SecurityServer documentation. The device may be a CryptoServer HSM (PCIe or LAN) device. The device line will follow one of these patterns, based on the HSM form-factor: Device = 288@<HSM IP address>
Hardware (LAN) HSM

OR

Device = /dev/cs2.0 Hardware (PCIe) HSM



To make your testing easier, it would be good to enable the PKCS#11 log file. That can be enabled by editing the **Logging Loglevel**. Set the **LogPath** and **Logging Loglevel** to 1. For testing you may want to increase it to 4.

The added **LogPath** points to a writable directory, not to a file.

If you encounter problems, check the log file named `cs_pkcs11_R3.log` in the **LogPath** defined directory. When you are done testing, you should change **Logging** to 1 or 2. This will limit the logging to only critical and important messages.

4.3 Create SO User and Initialize a Slot

You must initialize a slot with a custom label using p11tool2.

First using p11tool2 create, the SO or Security Officer and then using p11tool2 command initialize the slot that you want to use, and the slot user as shown below.

>_ Console

```
# p11tool2 slot=<slot_no> Label=<token_label> Login=ADMIN,ADMIN.key  
InitToken=ask  
  
# p11tool2 slot=<slot_no> LoginSO=ask InitPin=ask
```

5 Configuring OpenSSL to Use Utimaco SecurityServer

Bind9 uses openssl pkcs11 engine which is provided by libp11 package for signing the zone files. You can install libp11 with your existing openssl by following below steps:



If you want to install OpenSSL from source code, you can refer to OpenSSL integration guide available on Utimaco support portal.

5.1 Installing Libp11

1. Download the latest libp11 package from [Releases · OpenSC/libp11 · GitHub](#).

›_ Console

```
# wget https://github.com/OpenSC/libp11/releases/download/libp110.4.12/libp11-0.4.12.tar.gz
```

2. (Optional) It is recommended to update the system with latest security patch.

›_ Console

```
# dnf update
```

3. Extract the file.

›_ Console

```
# tar -xvf libp11-0.4.12.tar.gz
```

4. Go to libp11 directory, build, and install libp11 using the following commands.

> _ Console

```
# cd libp11-0.4.12

# ./configure prefix="/usr/local/libp11/"

# make

# make install

# export LD_LIBRARY_PATH=/usr/local/libp11/lib/:$LD_LIBRARY_PATH
```

5.2 Setting up Utimaco SecurityServer Library in OpenSSL Configuration File

1. Open the file `/etc/pki/tls/openssl.cnf` and enter the following line in the first line of the file.

openssl.cnf

```
openssl_conf = openssl_init
```

2. Enter the following lines under last section of `openssl.cnf` file.

openssl.cnf

```
[openssl_init]
engines=engine_section
[engine_section]
pkcs11 = pkcs11_section
[pkcs11_section]
engine_id = pkcs11
dynamic_path = /<engine_directory>/pkcs11.so
MODULE_PATH = /opt/utimaco/lib/libcs_pkcs11_R3.so
init = 0
```



Dynamic path and Module path will get changed according to the user environment.

5.2.1 Verifying PKCS#11 Engine

Run the command below to verify whether the OpenSSL Engine is available.

›_ Console

```
# openssl engine pkcs11 -t
```

```
[root@SP-Bind9 /]# openssl engine pkcs11 -t
(pkcs11) pkcs11 engine
      [ available ]
[root@SP-Bind9 /]#
```

Figure 1 : Verification of pkcs11 engine

6 Installing and Running Bind9

1. Download the latest version of source code file from [BIND 9 - ISC](#).
2. Extract the `tar` file.

›_ Console

```
# tar -xvf bind9-9.18.14.tar.gz
```

3. Go to the extracted folder and run `autoreconf` command.

›_ Console

```
# autoreconf -fi
```

4. Install the dependent packages for bind9.

›_ Console

```
# dnf install openssl-devel perl libicu-devel pkgconf userspace-rcu-devel  
libcap-devel libuv-devel
```

5. If you are using OpenSSL 1.1.1x run `configure` command as shown below.

›_ Console

```
# configure CC="gcc -m64" -enable-threads -disable-doh
```

```
[root@SP-Bind9-New bind9-9.18.14]# ./configure CC="gcc -m64" --enable-threads --disable-doh
configure: WARNING: unrecognized options: --enable-threads
checking build system type... x86_64-pc-linux-gnu
checking host system type... x86_64-pc-linux-gnu
checking target system type... x86_64-pc-linux-gnu
checking for a BSD-compatible install... /bin/install -c
checking whether build environment is sane... yes
checking for a thread-safe mkdir -p... /bin/mkdir -p
checking for gawk... gawk
checking whether make sets $(MAKE)... yes
checking whether make supports nested variables... yes
checking how to create a pax tar archive... gnutar
checking whether make supports nested variables... (cached) yes
checking whether to enable maintainer-specific portions of Makefiles... yes
checking whether make supports the include directive... yes (GNU style)
checking for gcc... gcc -m64
checking whether the C compiler works... yes
checking for C compiler default output file name... a.out
checking for suffix of executables...
checking whether we are cross compiling... no
checking for suffix of object files... o
checking whether we are using the GNU C compiler... yes
checking whether gcc -m64 accepts -g... yes
checking for gcc -m64 option to accept ISO C89... none needed
checking whether gcc -m64 understands -c and -o together... yes
checking dependency style of gcc -m64... gcc3
checking how to run the C preprocessor... gcc -m64 -E
checking for grep that handles long lines and -e... /bin/grep
checking for egrep... /bin/grep -E
checking for ANSI C header files... yes
checking for sys/types.h... yes
checking for sys/stat.h... yes
checking for stdlib.h... yes
```

Figure 2 : Configure command output

```
GeoIP2 access control (--enable-geoip)
DNS Response Policy Service interface (--enable-dnsrps)
Allow 'fixed' rrset-order (--enable-fixed-rrset)
Very verbose query trace logging (--enable-querytrace)
Single-query trace logging (--enable-singletrace)
CMocka Unit Testing Framework (--with-cmocka)
XML statistics (--with-libxml2)
JSON statistics (--with-json-c)
LMDB database to store configuration for 'addzone' zones (--with-lmdb)
IDN support (--with-libidn2)
-----
Configured paths:
prefix: /usr/local
sysconfdir: ${prefix}/etc
localstatedir: ${prefix}/var
-----
Compiler: gcc -m64
gcc (GCC) 8.5.0 20210514 (Red Hat 8.5.0-18)
Copyright (C) 2018 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

CFLAGS: -Wall -Wextra -Wwrite-strings -Wpointer-arith -Wno-missing-field-initializers -Wformat -Wshadow -Werror=implicit-funct
ion-declaration -Werror=missing-prototypes -Werror=format-security -Werror=parentheses -Werror=implicit -Werror=strict-prototy
pes -Werror=vla -fno-strict-aliasing -fno-delete-null-pointer-checks -fdiagnostics-show-option -g -O2 -pthread
CPPFLAGS: -D_FORTIFY_SOURCE=2
LDFLAGS:
-----
Unrecognized options:
--enable-threads
-----
For more detail, use --enable-full-report.
=====
```

Figure 3 : Configure command output

6. If you are using OpenSSL 3.0.x run `configure` command as shown below.

> _ Console

```
# configure CC="gcc -m64" CFLAGS=-DOPENSSL_API_COMPAT=10100 -enable-threads -  
disable-doh
```

```
[root@sp-BindRHEL9 bind9-9.18.14]# ./configure CC="gcc -m64" CFLAGS=-DOPENSSL_API_COMPAT=10100 -enable-threads --disable-doh  
configure: WARNING: unrecognized options: --enable-threads  
checking build system type... x86_64-pc-linux-gnu  
checking host system type... x86_64-pc-linux-gnu  
checking target system type... x86_64-pc-linux-gnu  
checking for a BSD-compatible install... /bin/install -c  
checking whether build environment is sane... yes  
checking for a thread-safe mkdir -p... /bin/mkdir -p  
checking for gawk... gawk  
checking whether make sets $(MAKE)... yes  
checking whether make supports nested variables... yes  
checking how to create a pax tar archive... gnutar  
checking whether make supports nested variables... (cached) yes  
checking whether to enable maintainer-specific portions of Makefiles... yes  
checking whether make supports the include directive... yes (GNU style)  
checking for gcc... gcc -m64  
checking whether the C compiler works... yes  
checking for C compiler default output file name... a.out  
checking for suffix of executables...  
checking whether we are cross compiling... no  
checking for suffix of object files... o  
checking whether we are using the GNU C compiler... yes  
checking whether gcc -m64 accepts -g... yes  
checking for gcc -m64 option to accept ISO C89... none needed  
checking whether gcc -m64 understands -c and -o together... yes  
checking dependency style of gcc -m64... gcc3  
checking how to run the C preprocessor... gcc -m64 -E  
checking for grep that handles long lines and -e... /bin/grep  
checking for egrep... /bin/grep -E  
checking for ANSI C header files... yes  
checking for sys/types.h... yes  
checking for sys/stat.h... yes  
checking for stdlib.h... yes
```

Figure 4 : Configuring Bind9

```

GSS-API (--with-gssapi)
DNS Response Policy Service interface (--enable-dnsrps)
Allow 'fixed' rrsset-order (--enable-fixed-rrset)
Very verbose query trace logging (--enable-querytrace)
Single-query trace logging (--enable-singletrace)
CMocka Unit Testing Framework (--with-cmocka)
XML statistics (--with-libxml2)
JSON statistics (--with-json-c)
LMDB database to store configuration for 'addzone' zones (--with-lmdb)
IDN support (--with-libidn2)
-----
Configured paths:
prefix: /usr/local
sysconfdir: ${prefix}/etc
localstatedir: ${prefix}/var
-----
Compiler: gcc -m64
gcc (GCC) 11.3.1 20221121 (Red Hat 11.3.1-4)
Copyright (C) 2021 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

CFLAGS: -Wall -Wextra -Wwrite-strings -Wpointer-arith -Wno-missing-field-initializers -Wformat -Wshadow -Werror=implicit-funct
ion-declaration -Werror=missing-prototypes -Werror=format-security -Werror=parentheses -Werror=implicit -Werror=strict-prototy
pes -Werror=vla -fno-strict-aliasing -fno-delete-null-pointer-checks -fdiagnostics-show-option -DOPENSSL_API_COMPAT=10100 -pth
read
CPPFLAGS: -D_FORTIFY_SOURCE=2
LDFLAGS:
-----
Unrecognized options:
--enable-threads
-----
For more detail, use --enable-full-report.

```

Figure 5 : Configuring Bind9



Bind9 by default doesn't support dynamic engine in OpenSSL 3.0.x. You need to run above command to support dynamic engine.

7. Build and install Bind9 as shown below.

_ Console

```

# make

# make install

```



It will install bind9 on default directory as `/usr/local/`.

```
[root@SP-Bind9 bind9-9.18.14]# make
make all-recursive
make[1]: Entering directory '/home/          /bind9-9.18.14'
Making all in .
make[2]: Entering directory '/home/          /bind9-9.18.14'
make[2]: Leaving directory '/home/          /bind9-9.18.14'
Making all in lib
make[2]: Entering directory '/home/          /bind9-9.18.14/lib'
Making all in isc
make[3]: Entering directory '/home/          /bind9-9.18.14/lib/isc'
CC      netmgr/libisc_la-tcp.lo
CC      netmgr/libisc_la-tcpdns.lo
CC      netmgr/libisc_la-timer.lo
CC      netmgr/libisc_la-tlsdns.lo
CC      netmgr/libisc_la-udp.lo
CC      netmgr/libisc_la-uv-compat.lo
CC      netmgr/libisc_la-uverr2result.lo
CC      libisc_la-aes.lo
CC      libisc_la-app.lo
CC      libisc_la-assertions.lo
CC      libisc_la-astack.lo
CC      libisc_la-backtrace.lo
CC      libisc_la-base32.lo
CC      libisc_la-base64.lo
CC      libisc_la-buffer.lo
CC      libisc_la-commandline.lo
CC      libisc_la-condition.lo
CC      libisc_la-counter.lo
CC      libisc_la-crc64.lo
CC      libisc_la-dir.lo
CC      libisc_la-entropy.lo
CC      libisc_la-errno.lo
CC      libisc_la-errno2result.lo
```

Figure 6 : Make output

```
[root@SP-Bind9 bind9-9.18.14]# make install
make install-recursive
make[1]: Entering directory '/home/          /bind9-9.18.14'
Making install in .
make[2]: Entering directory '/home/          /bind9-9.18.14'
make[3]: Entering directory '/home/          /bind9-9.18.14'
/bin/mkdir -p '/usr/local/etc'
/bin/install -c -m 644 bind.keys '/usr/local/etc'
make[3]: Nothing to be done for 'install-data-am'.
make[3]: Leaving directory '/home/          /bind9-9.18.14'
make[2]: Leaving directory '/home/          /bind9-9.18.14'
Making install in lib
make[2]: Entering directory '/home/          /bind9-9.18.14/lib'
Making install in isc
make[3]: Entering directory '/home/          /bind9-9.18.14/lib/isc'
make[4]: Entering directory '/home/          /bind9-9.18.14/lib/isc'
/bin/mkdir -p '/usr/local/lib'
/bin/sh ../libtool --mode=install /bin/install -c libisc.la '/usr/local/lib'
libtool: install: /bin/install -c .libs/libisc-9.18.14.so /usr/local/lib/libisc-9.18.14.so
libtool: install: (cd /usr/local/lib && { ln -s -f libisc-9.18.14.so libisc.so || { rm -f libisc.so && ln -s libisc-9.18.14.so
libisc.so; }; })
libtool: install: /bin/install -c .libs/libisc.lai /usr/local/lib/libisc.la
libtool: finish: PATH="/usr/share/Modules/bin:/sbin:/bin:/usr/sbin:/usr/bin:/sbin" ldconfig -n /usr/local/lib
-----
Libraries have been installed in:
  /usr/local/lib

If you ever happen to want to link against installed libraries
in a given directory, LIBDIR, you must either use libtool, and
specify the full pathname of the library, or use the '-LLIBDIR'
flag during linking and do at least one of the following:
- add LIBDIR to the 'LD_LIBRARY_PATH' environment variable
  during execution
```

Figure 7 : Make install output

8. Move the file `named.root.key` and `named.rfc1912.zones` to `/etc` directory.
9. Check the version of Bind9 as below.

```
[root@sp-BindRHEL9 named]# /usr/local/sbin/named -version
BIND 9.18.14 (Extended Support Version) <id:>
[root@sp-BindRHEL9 named]#
```

Figure 8 : Bind9 version output

10. Create a forward and reverse zone file at `/var/named` directory.

For example, below are samples files `example.net` and `exampleRev.net` for forward and reverse zone respectively.

example.net

```
# vi /var/named/example.net
$TTL 1D
$ORIGIN example.net.
@ IN SOA example.net. hostmaster.example.net. (
    0 ; serial
    1D ; refresh
    1H ; retry
    1W ; expire
    3H ) ; minimum
NS @
A 127.0.0.1
test IN A 172.23.0.69
mail IN A 172.23.0.18
neo IN A 172.23.0.15
AAAA ::1
```

exampleRev.net

```
Vi /var/named/exampleRev.net
$TTL 86400
@ IN SOA exampleRev.net. hostmaster.exampleRev.net.(
    2009072852 ; Serial
    28800 ; Refresh
    14400 ; Retry
    3600000 ; Expire
    86400 ) ; Minimum
    IN NS RHEL04.exampleRev.net.
    104 IN PTR RHEL04.exampleRev.net.
    105 IN PTR RHEL05.exampleRev.net.
    69 IN PTR fzrxqf3b03gu3cmgnrl2ry4xad.rx.internal.cloudapp.net
```

11. Make the appropriate changes in configuration file of bind9 that is `/usr/local/etc/named.conf` related to zone as highlighted.

Below is sample `named.conf` file.

named.conf

```
# vi /usr/local/etc/named.conf
options {
    listen-on port 53 { 127.0.0.1; 172.23.0.69; };
    listen-on-v6 port 53 { ::1; };
    directory "/var/named";
    dump-file "/var/named/data/cache_dump.db";
    statistics-file "/var/named/data/named_stats.txt";
    memstatistics-file "/var/named/data/named_mem_stats.txt";
    secroots-file "/var/named/data/named.secroots";
    recursing-file "/var/named/data/named.recursing";
    allow-query { localhost; };
    recursion no;
    dnssec-validation yes;
    managed-keys-directory "/var/named/dynamic";
    pid-file "/run/named/named.pid";
    session-keyfile "/run/named/session.key";
    include "/etc/crypto-policies/back-ends/bind.config";
};
logging {
    channel default_debug {
        file "data/named.run";
        severity dynamic;
    };
};
zone "." IN {
    type hint;
    file "named.ca";
};
zone "example.net" {
    type primary;
    file "example.net";
};
include "/etc/named.rfc1912.zones";
include "/etc/named.root.key";
```

12. Start the named service for bind9 using below command.

> Console

```
# /usr/local/sbin/named -f -c /usr/local/etc/named.conf
```

```
[root@SP-Bind9 /]# /usr/local/sbin/named -f -c /usr/local/etc/named.conf
```

Figure 9 : Start named service

13. Now from other terminal check that the named is running on port 53 using `netstat` command.

```
[root@SP-Bind9 tmp]# netstat -tunlp | grep named
tcp        0      0 172.23.0.69:53      0.0.0.0:*           LISTEN      7539/named
tcp        0      0 127.0.0.1:53        0.0.0.0:*           LISTEN      7539/named
tcp        0      0 127.0.0.1:953       0.0.0.0:*           LISTEN      7539/named
udp        0      0 172.23.0.69:53      0.0.0.0:*           7539/named
udp        0      0 172.23.0.69:53      0.0.0.0:*           7539/named
udp        0      0 127.0.0.1:53        0.0.0.0:*           7539/named
udp        0      0 127.0.0.1:53        0.0.0.0:*           7539/named
[root@SP-Bind9 tmp]#
```

Figure 10 : Check port of named services

7 Integrating Bind9 with CryptoServer HSM

This chapter includes the following sections:

- [Generate ZSK and KSK on CryptoServer HSM](#)
- [Generate Reference Key files of HSM Keys for Bind9](#)
- [Sign and Verify the Zone File](#)
- [Configuring Bind9 to Use Signed Zone File](#)
- [Key Rollover for KSK and ZSK](#)

7.1 Generate ZSK and KSK on CryptoServer HSM

To generate a zone-signing key (ZSK) and a key-signing key (KSK) on the HSM use the `p11tool2` utility. After the keys are generated on HSM, use `dnssec-keyfromlabel` provided by BIND to generate the key files for BIND containing a public key and an identifier of the actual private key.

7.1.1 Generating KSK and ZSK with RSA Algorithm

1. Run the following commands to generate a zone-signing key and a key-signing key using RSA algorithm:

›_ Console

```
# p11tool2 slot=0 LoginUser=ask PubKeyAttr=CKA_LABEL="ksk"  
PrvKeyAttr=CKA_LABEL="ksk" GenerateKeyPair=RSA  
  
# p11tool2 slot=0 LoginUser=ask PubKeyAttr=CKA_LABEL="zsk"  
PrvKeyAttr=CKA_LABEL="zsk" GenerateKeyPair=RSA
```

```
[root@SP-Bind9 bin]# ./p11tool2 slot=0 LoginUser=ask PubKeyAttr=CKA_LABEL="ksk" PrvKeyAttr=CKA_LABEL="ksk" GenerateKeyPair=RSA  
Enter normal user PIN:  
[root@SP-Bind9 bin]#
```

Figure 11 : Key generation output for KSK

```
[root@SP-Bind9 bin]# ./p11tool2 slot=0 LoginUser=ask PubKeyAttr=CKA_LABEL="zsk" PrvKeyAttr=CKA_LABEL="zsk" GenerateKeyPair=RSA
Enter normal user PIN:
[root@SP-Bind9 bin]#
```

Figure 12 : Key generation output for ZSK



Each key should have unique label that will be used to refer the private key in next steps.

2. List the keys using p11tool2 command.

›_ Console

```
# p11tool2 slot=0 LoginUser=ask ListObjects
```

```
[root@SP-Bind9 bin]# ./p11tool2 slot=0 LoginUser=ask ListObjects
Enter normal user PIN:

CKO_PUBLIC_KEY:
+ 1.1
  CKA_KEY_TYPE           = CKK_RSA
  CKA_UNIQUE_ID          = 5E1B9813-FD7A-4053-9778-4DFF1FDDEED7
  CKA_LABEL              = zsk
  CKA_ID                 =
+ 1.2
  CKA_KEY_TYPE           = CKK_RSA
  CKA_UNIQUE_ID          = F51E70F5-ABC2-4909-9215-3BB787C614C7
  CKA_LABEL              = ksk
  CKA_ID                 =

CKO_PRIVATE_KEY:
+ 2.1
  CKA_KEY_TYPE           = CKK_RSA
  CKA_UNIQUE_ID          = E09933F1-C278-4F76-9EC8-22E75E4145BD
  CKA_SENSITIVE          = CK_TRUE
  CKA_EXTRACTABLE        = CK_FALSE
  CKA_LABEL              = zsk
  CKA_ID                 =
```

Figure 13 : List key output

```
+ 2.2
  CKA_KEY_TYPE           = CKK_RSA
  CKA_UNIQUE_ID          = 37BC0A61-CC94-49EE-8BBF-9CF2EB092B9C
  CKA_SENSITIVE          = CK_TRUE
  CKA_EXTRACTABLE        = CK_FALSE
  CKA_LABEL              = ksk
  CKA_ID                 =
[root@SP-Bind9 bin]#
```

Figure 14 : List key output

7.1.2 Generating KSK and ZSK with ECC Algorithm

1. Run the following commands to generate a zone-signing key and a key-signing key using ECC:



A note on generating ECDSA keys: there is a bug in libp11 when looking up a key, that function compares keys only on their ID, not the label. So, when looking up a key it returns the first key, rather than the matching key. The workaround for this is when creating ECDSA keys, you should specify a unique ID:

_ Console

```
# ksk=$(echo "ksk" | openssl sha1 -r | awk '{print $1}')
# echo $ksk

# ./p11tool2 slot=0 LoginUser=ask PubKeyAttr=CKA_LABEL="ksk",CKA_ID=0x
a01834fcaa9401cefc8eeac928de96521cbd671a PrvKeyAttr=CKA_LABEL="ksk",CKA_ID=0x
a01834fcaa9401cefc8eeac928de96521cbd671a GenerateKeyPair=ECC

# zsk=$(echo "zsk" | openssl sha1 -r | awk '{print $1}')
# echo $zsk

# p11tool2 slot=0 LoginUser=ask

PubKeyAttr=CKA_LABEL="zsk",CKA_ID=0x67700db641c898be1b44a41d3fa68a7e0f33452b
PrvKeyAttr=CKA_LABEL="zsk",CKA_ID=0x67700db641c898be1b44a41d3fa68a7e0f33452b
GenerateKeyPair=ECC
```

```
[root@sp-BindRHEL9 bin]# ksk=$(echo "ksk" | openssl sha1 -r | awk '{print $1}')
[root@sp-BindRHEL9 bin]# echo $ksk
a01834fcaa9401cefc8eeac928de96521cbd671a
[root@sp-BindRHEL9 bin]#
```

Figure 15 : Key generation output for KSK

```
[root@sp-BindRHEL9 bin]# ./p11tool2 slot=0 LoginUser=ask PubKeyAttr=CKA_LABEL="ksk",CKA_ID=0xa01834fcaa9401cefc8eeac928de96521
cbd671a PrvKeyAttr=CKA_LABEL="ksk",CKA_ID=0xa01834fcaa9401cefc8eeac928de96521cbd671a GenerateKeyPair=ECC
Enter normal user PIN:
```

Figure 16 : Key generation output for KSK

```
[root@sp-BindRHEL9 bin]# zsk=$(echo "zsk" | openssl sha1 -r | awk '{print $1}')
[root@sp-BindRHEL9 bin]# echo $zsk
67700db641c898be1b44a41d3fa68a7e0f33452b
[root@sp-BindRHEL9 bin]#
```

Figure 17 : Key generation output for ZSK

```
[root@sp-BindRHEL9 bin]# ./p11tool2 slot=0 LoginUser=ask PubKeyAttr=CKA_LABEL="zsk",CKA_ID=0x67700db641c898be1b44a41d3fa68a7e0f33452b PrvKeyAttr=CKA_LABEL="zsk",CKA_ID=0x67700db641c898be1b44a41d3fa68a7e0f33452b GenerateKeyPair=ECC
Enter normal user PIN:
[root@sp-BindRHEL9 bin]#
```

Figure 18 : Key generation output for ZSK

2. List the keys using `p11tool2` command.

› _ Console

```
# p11tool2 slot=0 LoginUser=ask ListObjects
```

```
+ 1.5
CKA_KEY_TYPE           = CKK_ECDSA
CKA_UNIQUE_ID          = 186AA605-6B53-4B11-88BD-781A8D8A4184
CKA_LABEL              = zsk
CKA_ID                 =
                        0x67700DB6 41C898BE 1B44A41D 3FA68A7E | gp A D ? ~ |
                        0F33452B          | 3E+      |

+ 1.6
CKA_KEY_TYPE           = CKK_ECDSA
CKA_UNIQUE_ID          = 504EEB3C-54CF-48F9-BD23-BE54826D03EA
CKA_LABEL              = ksk
CKA_ID                 =
                        0x1F441E9C C3CEDC69 00F2089F 540C51BB | D i T Q |
                        81E41E15          |          |
```

Figure 19 : List key output

```

+ 2.4
  CKA_KEY_TYPE           = CKK_ECDSA
  CKA_UNIQUE_ID          = DA424FAB-2BFF-404F-928F-E02C722A6204
  CKA_SENSITIVE          = CK_TRUE
  CKA_EXTRACTABLE        = CK_FALSE
  CKA_LABEL              = zsk
  CKA_ID                 =
                        0x67700DB6 41C898BE 1B44A41D 3FA68A7E | gp A D ? ~ |
                        0F33452B      | 3E+      |

+ 2.5
  CKA_KEY_TYPE           = CKK_ECDSA
  CKA_UNIQUE_ID          = 6B486117-FFF9-4807-AEA8-CCC4ECA78E88
  CKA_SENSITIVE          = CK_TRUE
  CKA_EXTRACTABLE        = CK_FALSE
  CKA_LABEL              = ksk
  CKA_ID                 =
                        0x1F441E9C C3CEDC69 00F2089F 540C51BB | D i T Q |
                        81E41E15      |      |

```

Figure 20 : List key output



ECC keys which are generated above have default NIST-P256 curve.

7.2 Generate Reference Key files of HSM Keys for Bind9

Convert the RSA keys stored in the HSM into a format that BIND 9 understands. The [dnssec-keyfromlabel](#) tool from BIND 9 can link the raw keys stored in the HSM with the `K<zone>+<alg>+<id>` files. Provide the OpenSSL engine name (pkcs11), the algorithm and the PKCS#11 label that specify the token, the name of the PKCS#11 object (called label when generating the keys using p11tool2) and the HSM Slot PIN.

7.2.1 For RSA Keys

1. Generate key file for KSK.

> _ Console

```
# dnssec-keyfromlabel -E pkcs11 -f KSK -a RSASHA256 -l  
"pkcs11:token=Bind;object=ksk" example.net
```

Where the parameters:

- -E Is for engine.
- -l specifies the key label in pkcs11 URI format.
- -f specifies the key flag.
- -a is the algorithm.
- example.net is name of zone.

```
[root@SP-Bind9 bin]# ./dnssec-keyfromlabel -E pkcs11 -f KSK -a RSASHA256 -l "pkcs11:token=Bind;object=ksk" example.net  
Enter PKCS#11 token PIN for Bind:  
Kexample.net.+008+06500  
[root@SP-Bind9 bin]#
```

Figure 21 : Key file generation for KSK key

2. Generate key file for ZSK.

> _ Console

```
# dnssec-keyfromlabel -E pkcs11 -a RSASHA256 -l "pkcs11:token=Bind;object=zsk"  
example.net
```

```
[root@SP-Bind9 bin]# ./dnssec-keyfromlabel -E pkcs11 -a RSASHA256 -l "pkcs11:token=Bind;object=zsk" example.net  
Enter PKCS#11 token PIN for Bind:  
Kexample.net.+008+44509  
[root@SP-Bind9 bin]#
```

Figure 22 : Key file generation for ZSK key

Where the parameters:

- -E Is for engine.
- -l specifies the key label in pkcs11 URI format.

- -a is the algorithm.
- example.net is name of zone.

3. Verify that you have two KSK and two ZSK key files available.

›_ Console

```
# ls -l K*
```

```
-rw-r--r--. 1 root root 604 Jun 13 07:36 Kexample.net.+008+06500.key
-rw-----. 1 root root 591 Jun 13 07:36 Kexample.net.+008+06500.private
-rw-r--r--. 1 root root 606 Jun 13 07:37 Kexample.net.+008+44509.key
-rw-----. 1 root root 591 Jun 13 07:37 Kexample.net.+008+44509.private
```

Figure 23 : List files

7.2.2 For ECC Keys

1. Generate key file for KSK.

›_ Console

```
# dnssec-keyfromlabel -E pkcs11 -f KSK a- ECDSAP256SHA256 -l
"pkcs11:token=Bind;object=ksk" exampleecc.net
```

Where the parameters:

- -E is for engine.
- -l specifies the key label in pkcs11 URI format.
- -f specifies the key flag.
- -a is the algorithm.
- exampleecc.net is name of zone.

```
[root@sp-BindRHEL9 bin]# /usr/local/bin/dnssec-keyfromlabel -E pkcs11 -f KSK -a ECDSAP256SHA256 -l "pkcs11:token=Bind;object=k  
sk" exampleecc.net  
Enter PKCS#11 token PIN for Bind:  
Kexampleecc.net.+013+09870
```

Figure 24 : Key file generation for KSK key

2. Generate key file for ZSK.

> _ Console

```
# dnssec-keyfromlabel -E pkcs11 -a ECDSAP256SHA256 -l  
"pkcs11:token=Bind;object=zsk" exampleecc.net
```

```
[root@sp-BindRHEL9 bin]# /usr/local/bin/dnssec-keyfromlabel -E pkcs11 -a ECDSAP256SHA256 -l "pkcs11:token=Bind;object=zsk" exa  
mpleecc.net  
Enter PKCS#11 token PIN for Bind:  
Kexampleecc.net.+013+33330  
[root@sp-BindRHEL9 bin]#
```

Figure 25 : Key file generation for ZSK key

Where the parameters:

- -E Is for engine.
- -l specifies the key label in pkcs11 URI format.
- -f specifies the key flag.
- -a is the algorithm.
- example.net is name of zone.

3. Verify that you have two KSK and two ZSK key files available.

> _ Console

```
# ls -l K*
```

```
[root@sp-BindRHEL9 bin]# ls -l Kexampleecc.net*
-rwxrwxrwx. 1 root root 346 Jul 20 12:14 Kexampleecc.net.+013+09870.key
-rwxrwxrwx. 1 root root 212 Jul 20 12:14 Kexampleecc.net.+013+09870.private
-rwxrwxrwx. 1 root root 348 Jul 21 04:00 Kexampleecc.net.+013+33330.key
-rwxrwxrwx. 1 root root 212 Jul 21 04:00 Kexampleecc.net.+013+33330.private
-rwxrwxrwx. 1 root root 348 Jul 21 04:06 Kexampleecc.net.+013+50857.key
-rwxrwxrwx. 1 root root 216 Jul 21 04:06 Kexampleecc.net.+013+50857.private
-rwxrwxrwx. 1 root root 347 Jul 21 04:02 Kexampleecc.net.+013+52940.key
-rwxrwxrwx. 1 root root 216 Jul 21 04:02 Kexampleecc.net.+013+52940.private
[root@sp-BindRHEL9 bin]#
```

Figure 26 : List files

7.3 Sign and Verify the Zone File

During zone signing you need to provide the name of the OpenSSL engine using the -E command line option.

7.3.1 Sign and Verify Zone Files with RSA Keys

1. Use `dnssec-signzone` command to perform zone signing.

Console

```
# dnssec-signzone -E pkcs11 -S -o example.net /var/named/example.net
```

```
[root@SP-Bind9 bin]# /usr/local/bin/dnssec-signzone -E pkcs11 -S -o example.net /var/named/example.net
Fetching example.net/RSASHA256/6500 (KSK) from key repository.
Fetching example.net/RSASHA256/44509 (ZSK) from key repository.
Verifying the zone using the following algorithms:
- RSASHA256
Zone fully signed:
Algorithm: RSASHA256: KSKs: 1 active, 0 stand-by, 0 revoked
                ZSKs: 1 active, 0 stand-by, 0 revoked
/var/named/example.net.signed
```

Figure 27 : Zone signing for RSA key

This generates the `example.net.signed` file.

2. Verify the signed zone file.

_ Console

```
# dnssec-verify -E pkcs11 -z -o example.net /var/named/example.net.signed  
Kexample.net.+008+06500.key Kexample.net.+008+44509.key
```

```
[root@SP-Bind9 bin]# /usr/local/bin/dnssec-verify -E pkcs11 -z -o example.net /var/named/example.net.signed Kexample.net.+008+06500.key Kexample.net.+008+44509.key  
Loading zone 'example.net' from file '/var/named/example.net.signed'  
  
Verifying the zone using the following algorithms:  
- RSASHA256  
Zone fully signed:  
Algorithm: RSASHA256: KSKs: 1 active, 0 stand-by, 0 revoked  
ZSKs: 1 active, 0 stand-by, 0 revoked
```

Figure 28 : Zone verification for RSA key

Where `Kexample.net.+008+06500.key` is key file generated for ksk key and `Kexample.net.+008+44509.key` is key file generated for zsk key.

7.3.2 Sign and Verify Zone Files with ECC Keys

1. Use `dnssec-signzone` command below to perform zone signing.

_ Console

```
# dnssec-signzone -E pkcs11 -S -o exampleecc.net /var/named/exampleecc.net
```

```
[root@sp-BindRHEL9 bin]# /usr/local/bin/dnssec-signzone -E pkcs11 -S -o exampleecc.net /var/named/exampleecc.net  
Enter PKCS#11 token PIN for Bind:  
Fetching exampleecc.net/ECDSAP256SHA256/8823 (KSK) from key repository.  
Fetching exampleecc.net/ECDSAP256SHA256/45889 (ZSK) from key repository.  
Verifying the zone using the following algorithms:  
- ECDSAP256SHA256  
Zone fully signed:  
Algorithm: ECDSAP256SHA256: KSKs: 1 active, 0 stand-by, 0 revoked  
ZSKs: 1 active, 0 stand-by, 0 revoked  
/var/named/exampleecc.net.signed  
[root@sp-BindRHEL9 bin]#
```

Figure 29 : Zone signing for ECC key

This generates the `exampleecc.net.signed` file.

2. Verify the signed zone file.

›_ Console

```
# dnssec-verify -E pkcs11 -o exampleecc.net /var/named/exampleecc.net.signed
```

```
[root@sp-BindRHEL9 bin]# /usr/local/bin/dnssec-verify -E pkcs11 -o exampleecc.net /var/named/exampleecc.net.signed Kexampleecc.net.+013+08823.key Kexampleecc.net.+013+45889.key
Loading zone 'exampleecc.net' from file '/var/named/exampleecc.net.signed'

Verifying the zone using the following algorithms:
- ECDSAP256SHA256
Zone fully signed:
Algorithm: ECDSAP256SHA256: KSKs: 1 active, 0 stand-by, 0 revoked
                        ZSKs: 1 active, 0 stand-by, 0 revoked
[root@sp-BindRHEL9 bin]#
```

Figure 30 : Zone verification for ECC key

7.4 Configuring Bind9 to Use Signed Zone File

1. Once the zone signing is done, you need to make changes in configuration file `named.conf` as below:

named.conf

```
# vi /usr/local/etc/named.conf
...
dnssec-validation yes;
...
zone "example.net" {
    type primary;
    file "example.net.signed";
};
```

2. Stop and start the bind9 service using below command.

›_ Console

```
# /usr/local/sbin/named -f -4 -E pkcs11 -c /usr/local/etc/named.conf
```

3. Verify log file `/var/log/messages` shows the signed zone is loaded successfully.

›_ Console

```
"zone example.net/IN: loaded serial 0 (DNSSEC signed)
```

```
Jun 13 09:06:57 SP-Bind9 named[25122]: all zones loaded Jun 13 09:06:57 SP-Bind9  
named[25122]: running"
```

```
Jun 16 05:46:34 SP-Bind9 named[8619]: zone localhost.localdomain/IN: loaded serial 0  
Jun 16 05:46:34 SP-Bind9 named[8619]: zone localhost/IN: loaded serial 0  
Jun 16 05:46:34 SP-Bind9 named[8619]: zone examplecc.net/IN: loaded serial 0 (DNSSEC signed)  
Jun 16 05:46:34 SP-Bind9 named[8619]: all zones loaded  
Jun 16 05:46:34 SP-Bind9 named[8619]: running  
Jun 16 05:46:34 SP-Bind9 named[8619]: managed-keys-zone: Key 20326 for zone . is now trusted (acceptance timer complete)  
Jun 16 05:46:34 SP-Bind9 named[8619]: resolver priming query complete: success
```

Figure 31 : Log output

7.5 Key Rollover for KSK and ZSK

7.5.1 Generate New KSK and ZSK

1. Generate new key pair as ksk1 and zks1 on HSM as below.

> _ Console

```
# p11tool2 slot=0 LoginUser=ask PubKeyAttr=CKA_LABEL="ksk1"  
PrvKeyAttr=CKA_LABEL="ksk1" GenerateKeyPair=RSA  
  
# p11tool2 slot=0 LoginUser=ask PubKeyAttr=CKA_LABEL="zsk1"  
PrvKeyAttr=CKA_LABEL="zsk1" GenerateKeyPair=RSA
```



If you want to generate new ECC keys, follow the steps in section *Generating KSK and ZSK with ECC Algorithm*.

7.5.2 KSK Rollover

1. Generate key file for new KSK.

> _ Console

```
# dnssec-keyfromlabel -E pkcs11 -a RSASHA256 -f KSK -l  
"pkcs11:token=Bind;object=ksk1" example.net
```

```
[root@SP-Bind9 bin]# /usr/local/bin/dnssec-keyfromlabel -E pkcs11 -a RSASHA256 -f KSK -l "pkcs11:token=Bind;object=ksk1" example.net  
Kexample.net.+008+33262
```

Figure 32 : Key file for new KSK



If you want to generate key files for ECC keys, follow steps in section *For ECC Keys*.

2. Add the new KSK to the zone file `example.net`.

example.net

```
...
$include "/usr/local/bin/Kexample.net.+008+65395.key"; //KSK old
$include "/usr/local/bin/Kexample.net.+008+33262.key"; //KSK new
...
```

3. Sign the zone with the old and new KSK.

> _ Console

```
# dnssec-signzone -E pkcs11 -x -o example.net -k Kexample.net.+008+65395 -k
Kexample.net.+008+33262 /var/named/example.net
```

```
[root@SP-Bind9 bin]# dnssec-signzone -E pkcs11 -x -o example.net -k Kexample.net.+008+65395 -k Kexample.net.+008+33262 /var/named/example.net
Verifying the zone using the following algorithms:
- RSASHA256
Zone fully signed:
Algorithm: RSASHA256: KSKs: 2 active, 0 stand-by, 0 revoked
                  ZSKs: 1 active, 0 present, 0 revoked
/var/named/example.net.signed
[root@SP-Bind9 bin]#
```

Figure 33 : Signing zone with old and new KSK

4. Wait for the zone transfer time, TTL of DNSKEY resource record set and TTL on the DS record set.
5. Remove the old KSK entry from zone file `example.net`.

example.net

```
...
$include "/usr/local/bin/Kexample.net.+008+33262.key"; //KSK new
...
```

6. Sign the zone with new KSK.

> _ Console

```
# dnssec-signzone -E pkcs11 -x -o example.net -k Kexample.net.+008+33262 /var/named/example.net
```

```
[root@SP-Bind9 bin]# dnssec-signzone -E pkcs11 -x -o example.net -k Kexample.net.+008+33262 /var/named/example.net
Verifying the zone using the following algorithms:
- RSASHA256
Zone fully signed:
Algorithm: RSASHA256: KSKs: 1 active, 0 stand-by, 0 revoked
                  ZSKs: 1 active, 0 present, 0 revoked
/var/named/example.net.signed
[root@SP-Bind9 bin]#
```

Figure 34 : Signing the zone with new KSK

7.5.3 ZSK Rollover

1. Generate key file for new ZSK.

> _ Console

```
# dnssec-keyfromlabel -E pkcs11 -a RSASHA256 -l
"pkcs11:token=Bind;object=zsk1" example.net
```

```
[root@SP-Bind9 bin]# /usr/local/bin/dnssec-keyfromlabel -E pkcs11 -a RSASHA256 -l "pkcs11:token=Bind;object=zsk1" example.net
Kexample.net.+008+33663
[root@SP-Bind9 bin]#
```

Figure 35 : Key file for new ZSK



If you want to generate key files for ECC keys, follow steps in section *For ECC Keys*.

2. Add the new ZSK to the zone file `example.net`.

example.net

```
...
$include "/usr/local/bin/Kexample.net.+008+60816.key"; //ZSK old
$include "/usr/local/bin/Kexample.net.+008+33663.key"; //ZSK new
...
```

3. Re-sign the zone with the KSK and old ZSK.

›_ Console

```
# dnssec-signzone -E pkcs11 -x -o example.net -k Kexample.net.+008+33262
/var/named/example.net Kexample.net.+008+60816
```

```
[root@SP-Bind9 bin]# dnssec-signzone -E pkcs11 -x -o example.net -k Kexample.net.+008+33262 /var/named/example.net Kexample.net.+008+60816
Verifying the zone using the following algorithms:
- RSASHA256
Zone fully signed:
Algorithm: RSASHA256: KSKs: 1 active, 0 stand-by, 0 revoked
                  ZSKs: 1 active, 1 present, 0 revoked
/var/named/example.net.signed
[root@SP-Bind9 bin]#
```

Figure 36 : Signing zone with old ZSK

4. Wait for the zone transfer time and TTL of the key set.
5. Sign the zone with new ZSK.

›_ Console

```
# dnssec-signzone -E pkcs11 -x -o example.net -k Kexample.net.+008+33262
/var/named/example.net Kexample.net.+008+33663
```

```
[root@SP-Bind9 bin]# dnssec-signzone -E pkcs11 -x -o example.net -k Kexample.net.+008+33262 /var/named/example.net Kexample.net.+008+33663
Verifying the zone using the following algorithms:
- RSASHA256
Zone fully signed:
Algorithm: RSASHA256: KSKs: 1 active, 0 stand-by, 0 revoked
                        ZSKs: 1 active, 1 present, 0 revoked
/var/named/example.net.signed
[root@SP-Bind9 bin]#
```

Figure 37 : Signing zone with new ZSK

6. Wait for the zone transfer time and maximum TTL used in the zone.
7. Remove old ZSK from the zone file example.net

example.net

```
...
$include "/usr/local/bin/Kexample.net.+008+33663.key"; //ZSK new
...
```

8. Re-sign the zone with the KSK. Now we have only one ZSK in `example.net` so it will automatically pick this new ZSK for signing zone.

_ Console

```
# dnssec-signzone -E pkcs11 -x -o example.net -k Kexample.net.+008+33262 /var/named/example.net
```

```
[root@SP-Bind9 bin]# dnssec-signzone -E pkcs11 -x -o example.net -k Kexample.net.+008+33262 /var/named/example.net
Verifying the zone using the following algorithms:
- RSASHA256
Zone fully signed:
Algorithm: RSASHA256: KSKs: 1 active, 0 stand-by, 0 revoked
                        ZSKs: 1 active, 0 present, 0 revoked
/var/named/example.net.signed
[root@SP-Bind9 bin]#
```

Figure 38 : Signing zone with new ZSK

9. Stop and start the named service using below command.

> _ Console

```
# /usr/local/sbin/named -f -4 -E pkcs11 -c /usr/local/etc/named.conf
```

```
[root@SP-Bind9 bin]# /usr/local/sbin/named -f -4 -E pkcs11 -c /usr/local/etc/named.conf
```

Figure 39 : Starting named service



This completes the Integration for Bind9 with Utimaco CryptoServer HSM.

8 Troubleshooting

Error	Diagnosis
CKR_USER_PIN_NOT_INITIALIZED	PKCS#11 Slot is not initialized.
CKR_PIN_INCORRECT	Check user PIN
error: zone example.net/IN (unsigned): loading from master file example.net failed: permission denied	Provide permission to zone file as <code>chmod -R 750 example.net</code> and change the ownership of file to user that is running the named service

Table 6: List of errors and their diagnoses

9 Further Information

This document forms a part of the information and support which is provided by Utimaco IS GmbH. Additional documentation can be found on the product CD in the Documentation directory.

All CryptoServer HSM product documentation is also available at the Utimaco IS GmbH website:
<https://utimaco.com/>.

10 References

Reference	Title/Company	Document No.
[CSTrSh]	CryptoServer Troubleshooting/Utimaco IS GmbH	M011-0008-en
[CSP11Tool2]	CryptoServer_p11tool2_Manual.pdf	2012-0004
[CSPKCSM]	CryptoServer - PKCS#11 P11CAT Manual	M013-0001-en
[CSLAN5]	CryptoServerLAN_Manual_Systemadministrators.pdf	2018-0004

Table 7: List of references

11 Contact and Support Information

You can reach us from Monday to Friday, 09.00 a.m. to 05.00 p.m., Central European Time (CET).

Utimaco IS GmbH
Krefelder Str. 220
52070 Aachen
Germany

RMA Query

If you need to send the device back to Utimaco IS GmbH, please open a new RMA case (Return Merchandise Authorization). We request that you use the following web address. RMA cases cannot be opened by email or phone.

<https://support.hsm.utimaco.com/support/rma/new>

Other Support Queries

- Mail (preferred contact method)
support@utimaco.com
Attach the diagnostic information to your email.
- Web portal
<https://support.hsm.utimaco.com/support/cases/new/>
The diagnostic information will be requested in our response if necessary.
- By phone
AMERICAS +1-844-UTIMACO (+1 844-884-6226)
EMEA +49 800-627-3081
APAC +81 800-919-1301
The diagnostic information will be requested in our response if necessary.