The GNU Privacy Guard Team

GnuPG

2.5.18

Integration Guide

u.trust GP HSM

6.4.0

utimaco®

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	1.0.0
Date	2026-06-12
Status	PUBLISHED
Document No.	IG-2026-0060

Table of Contents

1 Introduction

This Integration Guide describes the process of integrating GnuPG (GNU Privacy Guard) with the u.trust General Purpose (GP) Hardware Security Module (HSM). The integration enables secure cryptographic operations by leveraging the HSM as a trusted key storage and processing environment.

The u.trust GP HSM provides a secure and tamper-resistant platform for generating, storing, and managing cryptographic keys and certificates used by GnuPG. By integrating with the HSM, sensitive cryptographic material is protected from exposure, ensuring that operations such as encryption, decryption, digital signing, and signature verification are performed in a secure and compliant manner.

1.1 About This Guide

This guide provides a comprehensive overview of the integration between GnuPG and the u.trust GP HSM. It is intended to support users in understanding, configuring, and validating the integration in a secure and structured manner.

The document outlines the end-to-end process, starting from prerequisites and environment setup to detailed configuration steps and validation procedures.

1.2 Target Audience for This Guide

This guide is intended for GnuPG and Utimaco HSM administrators.

1.3 Purpose of the Integration

The purpose of this integration is to enhance the security of GnuPG-based cryptographic operations by leveraging the u.trust GP HSM as a secure key storage and processing platform. By offloading sensitive cryptographic functions to the HSM, private keys and certificates are protected within a tamper-resistant environment, reducing the risk of key exposure and unauthorized access. This ensures that critical operations such as encryption, decryption, digital signing, and verification are performed in a secure and controlled manner.

1.4 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
`Monospaced`	Code that is given for explanation or as an example, file paths	`certreq.exe -new request.inf IISCertRequest.csr`
Italic	References and important terms	Operating system listed in *Tested Versions*

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.

1.5 Abbreviations

The following abbreviations are used in this guide:

Abbreviation	Meaning
BSD	Berkeley Source Distribution

Abbreviation	Meaning
CSADM	CryptoServer Command-line Administration Tool
GnuPG (GPG)	GNU Privacy Guard
GUI	Graphical User Interface
GP HSM	General Purpose Hardware Security Module
PGP	Pretty Good Privacy
S/MIME	Secure/Multipurpose Internet Mail Extensions
SSH	Secure Shell or Secure Socket Shell
ID	Identifier or Identification
IP	Internet Protocol
LAN	Local Area Network
NPTH	New GNU Portable Threads
PCIe	PCI Express Interface
PIN	Personal Identification number
PKCS#11	Public-Key Cryptography Standard #11
RPM	Red Hat Package Manager

Abbreviation	Meaning
RSA	Rivest-Shamir-Adleman
SCD	Smart Card Daemon
SO	Security Officer
CA	Certificate Authority
URL	Uniform Resource Locator
CSR	Certificate Signing Request
PEM	Privacy-Enhanced Mail
DER	Distinguished Encoding Rules
libksba	Library Kasbah
libgcrypt	Library GNU Crypt
libassuan	Library Assuan
libgpg-error	Library GNU Privacy Guard Error
API	Application Programming Interface

Table 2: List of abbreviations

2 Overview

2.1 GnuPG (GPG)

GnuPG is a complete and free implementation of the OpenPGP standard as defined by RFC4880 (also known as PGP). GnuPG allows you to encrypt and sign your data and communications; it features a versatile key management system, along with access modules for all kinds of public key directories. GnuPG, also known as GPG, is a command line tool with features for easy integration with other applications. GnuPG also provides support for S/MIME and Secure Shell (ssh).

Gnupg-pkcs1-scd is a project to implement a BSD-licensed smart-card daemon to enable the use of PKCS#11 tokens with GnuPG.

2.2 Utimaco u.trust GP HSM

The u.trust GP HSM is a hardware security module developed by Utimaco IS GmbH. The u.trust GP 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.

3 Integration Requirements and Prerequisites

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

This guide assumes that the user has already installed and configured the required software.

3.1 Tested Versions

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

Operating System	GnuPG Version	GnuPG PKCS11 SCD Version	Utimaco Security Server Version	Utimaco HSM
Debian 12 Debian 13	2.5.18	0.11.0	SecurityServer V6.4.0	u.trust GP HSM CSe-Series/Se-Series

Table 3: List of tested versions

3.2 Software Requirements

Software	Software Requirements
GnuPG	GnuPG 2.5.18
GnuPG PKCS11 SCD	GnuPG PKCS11 SCD 0.11.0
HSM Interface	SecurityServer PKCS#11 Provider

Table 4: List of software requirements

3.3 Hardware Requirements

Hardware	Hardware Requirements
Utimaco LAN HSM	u.trust GP HSM CSe-Series/Se-Series LAN with firmware SecurityServer 6.4.0 or higher
Utimaco PCI-e HSM	u.trust GP HSM CSe-Series/Se-Series PCI-e with firmware SecurityServer 6.4.0 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

Please ensure that:

- The u.trust GP HSM is set up and configured. Refer to the u.trust GP HSM documentation to set up the HSM.
- The u.trust GP HSM Default Admin is replaced with a new admin user.
- The operating system used is listed in Tested Versions.
- The SecurityServer used is listed in Tested Versions.
- The public and private key pair has been created and stored onto each HSM. Refer to the u.trust GP HSM documentations to set up the keys.
- The PKCS#11 library is set up and configured as per the environment. Refer to the u.trust GP HSM documentations to set up and configure the PKCS#11 library for u.trust GP HSM.
- You familiarize yourself with the GnuPG. Refer to the GnuPG Documentation portal for more information.

4 Installing and Configuring Utimaco SecurtyServer Software

4.1 Downloading and Installing 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 to the appropriate location on the GnuPG Server.
2. Create a Utimaco folder under `/etc` directory and further create 2 directories: `/etc/utimaco/bin` and `/etc/utimaco/lib`.

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

3. Copy the pkcs11 library file `libcs_pkcs11_R3.so` from Utimaco GP HSM software to the `/etc/utimaco/lib` directory.

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

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

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

4.2 u.trust GP HSM PKCS#11 Configuration

1. Create the directory `/etc/utimaco/PKCS11`. 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 the `/etc/utimaco/PKCS11` directory.

```
# mkdir /etc/utimaco/PKCS11
```

```
# cd <install directory>/Software/Linux/x86-64/Crypto_APIs/PKCS11_R3/sample # cp
cs_pkcs11_R3.cfg /etc/utimaco/PKCS11 # cd /etc/utimaco/PKCS11
```

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

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

1 cs_pkcs11_R3.cfg

For more information regarding the commands and command parameters, please check the u.trust GP HSM documentation. The device may be a u.trust GP 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, you can 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 should initialize a slot with a custom label using `p11tool2`.

1. Using `p11tool2`, create the SO or Security Officer.

2. Using the `p11tool2` command, initialize the slot that you want to use, as well as the slot user, as shown below.

```
# ./p11tool2 slot=<slot_no> Label=<token_label> Login=ADMIN,ADMIN.key
InitToken=<SO_PIN>
# ./p11tool2 slot=0 LoginSO=<SO_PIN> InitPin=<CryptoUser_PIN>
```

This CryptoUser PIN is the slot PIN that will be used in this guide for every crypto operation.

5 Installing and Configuring GnuPG

5.1 Installing Dependent Packages for GnuPG

The GnuPG requires the following dependent packages:

- npth.
- libgpg-error.
- libgcrypt.
- libksba.
- libassuan.
- pinentry.

5.1.1 Installing Libgpg-error

1. Download the `libgpg-error` installation file.

```
# wget https://www.gnupg.org/ftp/gcrypt/libgpg-error/libgpg-error-1.59.tar.bz2
```

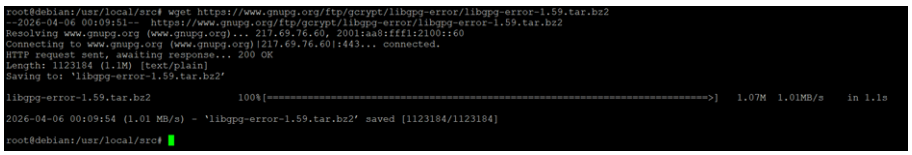

Figure 1 : Downloading libgpg-error

2. Extract the file.

```
# tar -xjf libgpg-error-1.59.tar.bz2
```

3. Go to the directory where the file is extracted.

```
# cd libgpg-error-1.59
```

4. Run the following command to compile and install.

```
# ./configure --prefix=/usr/local
# make
# make install
```

```
root@debian:/usr/local/src/libgpg-error-1.59# ./configure --prefix=/usr/local
checking for a BSD-compatible install... /usr/bin/install -c
checking whether build environment is sane... yes
checking for a race-free mkdir -p... /usr/bin/mkdir -p
checking for gawk... no
checking for mawk... mawk
checking whether make sets $(MAKE)... yes
checking whether make supports nested variables... yes
checking whether to enable maintainer-specific portions of Makefiles... no
checking whether make supports nested variables... (cached) yes
checking build system type... x86_64-pc-linux-gnu
checking host system type... x86_64-pc-linux-gnu
configure: autobuild project... libgpg-error
configure: autobuild revision... 1.59
configure: autobuild hostname... debian
configure: autobuild timestamp... 20260406T071311Z
checking for gcc... gcc
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 the compiler supports GNU C... yes
checking whether gcc accepts -g... yes
checking for gcc option to enable C11 features... none needed
checking whether gcc understands -c and -o together... yes
checking whether make supports the include directive... yes (GNU style)
checking dependency style of gcc... gcc3
checking how to run the C preprocessor... gcc -E
checking for gawk... (cached) mawk
checking for ar... ar
checking for stdio.h... yes
checking for stdlib.h... yes
checking for string.h... yes
checking for inttypes.h... yes
checking for stdint.h... yes
checking for strings.h... yes
checking for sys/stat.h... yes
checking for sys/types.h... yes
checking for unistd.h... yes
checking for wchar.h... yes
checking for minix/config.h... no
checking for threads.h... yes
checking whether it is safe to define __EXTENSIONS__... yes
checking whether _XOPEN_SOURCE should be defined... no
checking for nl_langinfo and CODESET... yes
checking for special C compiler options needed for large files... no
```

Figure 2 : Installing libgpg-error

```
checking for library containing socket... none required
checking whether readline via "-lreadline" is present and sane... no
checking whether readline via "-lreadline -ltermcap" is present and sane... no
checking whether readline via "-lreadline -lcurses" is present and sane... no
checking whether readline via "-lreadline -lncurses" is present and sane... no
checking for unsigned long long int... yes
configure: checking system features for estream-printf
checking for stdint.h... (cached) yes
checking for long long int... yes
checking for long double... yes
checking for intmax_t... yes
checking for uintmax_t... yes
checking for ptrdiff_t... yes
checking size of unsigned long... 8
checking size of void *... 8
checking for nl_langinfo and THOUSEP... yes
configure: checking system features for estream
checking for memrchr... yes
checking whether to enable log_clock... no
checking that generated files are newer than configure... done
configure: creating ./config.status
config.status: creating Makefile
config.status: creating doc/Makefile
config.status: creating po/Makefile.in
config.status: creating m4/Makefile
config.status: creating src/Makefile
config.status: creating tests/Makefile
config.status: creating lang/Makefile
config.status: creating lang/cl/Makefile
config.status: creating lang/cl/gpg-error.asd
config.status: creating src/versioninfo.rc
config.status: creating src/gpg-error.w32-manifest
config.status: creating src/gpg-error.pc
config.status: creating src/gpg-error-config-old
config.status: creating src/gpgrt-config
config.status: creating src/gpg-error-config-test.sh
config.status: creating config.h
config.status: executing depfiles commands
config.status: executing libtool commands
config.status: executing po-directories commands
config.status: creating po/POTFILES
config.status: creating po/Makefile

        libgpg-error v1.59 has been configured as follows:

        Revision: 3debf54  (15851)
        Platform: x86_64-pc-linux-gnu

root@debian:/usr/local/src/libgpg-error-1.59#
```

Figure 3 : Installing libgpg-error continued

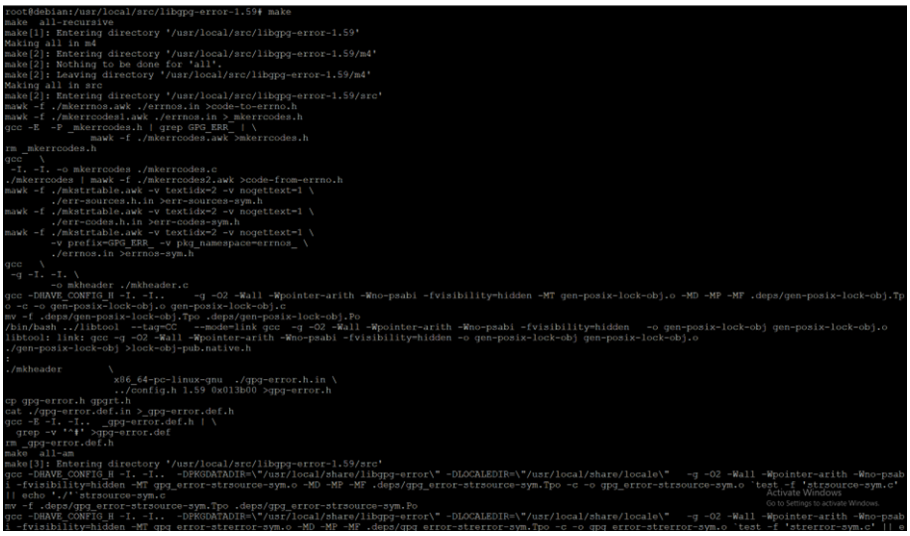

Figure 4 : Installing libgpg-error_make

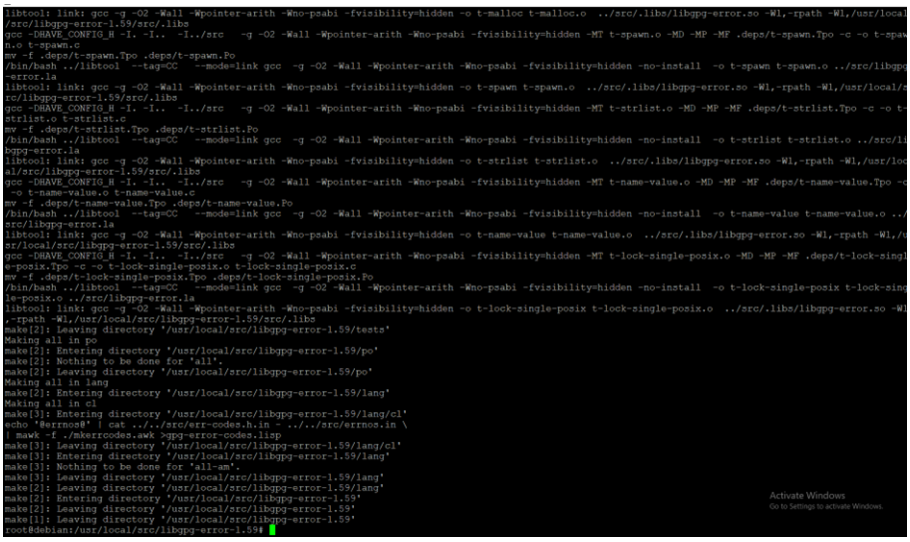

Figure 5 : Installing libgpg-error_make continued

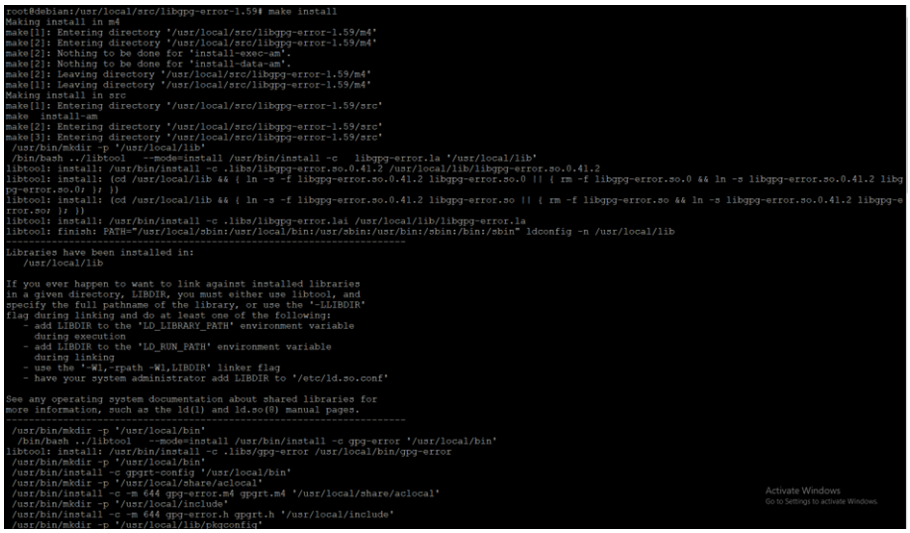

Figure 6 : Installing libgpg-error_make install

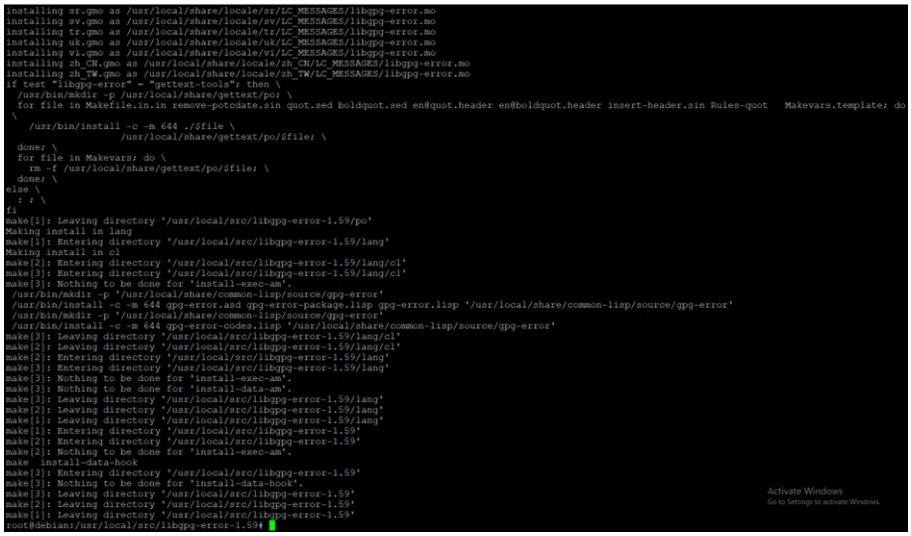

Figure 7 : Installing libgpg-error_make install continued

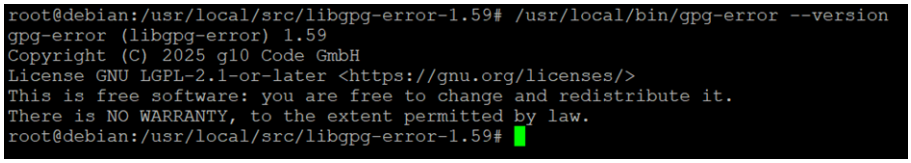

Figure 8 : Libgpg-error version

5.1.2 Installing Libgcrypt

1. Download the `libgcrypt` installation file.

```
# wget https://www.gnupg.org/ftp/gcrypt/libgcrypt/libgcrypt-1.11.0.tar.bz2
```

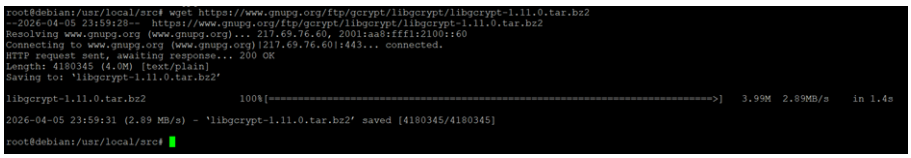

Figure 9 : Downloading libgcrypt

2. Extract the file.

```
# tar -xjf libgcrypt-1.11.0.tar.bz2
```

3. Go to the directory where the file is extracted.

```
# cd /usr/local/src/libgcrypt-1.11.0
```

4. Run the following command to compile and install.

```
# ./configure --prefix=/usr/local
# make
# make install
```

```
root@debian:/usr/local/src/libgpg-error-1.59# cd /usr/local/src/libgcrypt-1.11.0
root@debian:/usr/local/src/libgcrypt-1.11.0# ./configure --prefix=/usr/local
checking for a BSD-compatible install... /usr/bin/install -c
checking whether build environment is sane... yes
checking for a race-free mkdir -p... /usr/bin/mkdir -p
checking for gawk... no
checking for mawk... mawk
checking whether make sets $(MAKE)... yes
checking whether make supports nested variables... yes
checking build system type... x86_64-pc-linux-gnu
checking host system type... x86_64-pc-linux-gnu
checking whether to enable maintainer-specific portions of Makefiles... no
checking whether make supports nested variables... (cached) yes
checking whether make supports the include directive... yes (GNU style)
checking for gcc... gcc
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 the compiler supports GNU C... yes
checking whether gcc accepts -g... yes
checking for gcc option to enable C11 features... none needed
checking whether gcc understands -c and -o together... yes
checking dependency style of gcc... gcc3
checking for stdio.h... yes
checking for stdlib.h... yes
checking for string.h... yes
checking for inttypes.h... yes
checking for stdint.h... yes
checking for strings.h... yes
checking for sys/stat.h... yes
checking for sys/types.h... yes
checking for unistd.h... yes
checking for wchar.h... yes
checking for minix/config.h... no
checking whether it is safe to define __EXTENSIONS__... yes
checking whether _XOPEN_SOURCE should be defined... no
checking whether make sets $(MAKE)... (cached) yes
checking for gcc... (cached) gcc
checking whether the compiler supports GNU C... (cached) yes
checking whether gcc accepts -g... (cached) yes
checking for gcc option to enable C11 features... (cached) none needed
checking whether gcc understands -c and -o together... (cached) yes
checking dependency style of gcc... (cached) gcc3
checking how to run the C preprocessor... gcc -E
checking dependency style of gcc... gcc3
checking for library containing strerror... none required
```

Figure 10 : Installing libgcrypt

```
config.status: creating tests/Makefile
config.status: creating tests/hashtest-6g
config.status: creating tests/hashtest-256g
config.status: creating tests/basic-disable-all-hwf
config.status: creating config.h
config.status: linking mpi/amd64/mpih-add1.S to mpi/mpih-add1-asm.S
config.status: linking mpi/amd64/mpih-sub1.S to mpi/mpih-sub1-asm.S
config.status: linking mpi/amd64/mpih-mul1.S to mpi/mpih-mul1-asm.S
config.status: linking mpi/amd64/mpih-mul2.S to mpi/mpih-mul2-asm.S
config.status: linking mpi/amd64/mpih-mul3.S to mpi/mpih-mul3-asm.S
config.status: linking mpi/amd64/mpih-lshift.S to mpi/mpih-lshift-asm.S
config.status: linking mpi/amd64/mpih-rshift.S to mpi/mpih-rshift-asm.S
config.status: linking mpi/amd64/mpi-asm-defs.h to mpi/mpi-asm-defs.h
config.status: executing depfiles commands
config.status: executing libtool commands
config.status: executing gcrypt-conf commands

        Libgcrypt v1.11.0 has been configured as follows:

        Platform:                 GNU/Linux (x86_64-pc-linux-gnu)
        Hardware detection module: libgcrypt_la-hwf-x86
        Enabled cipher algorithms: arcfour blowfish cast5 des aes twofish
                                  serpent rfc2268 seed camellia idea salsa20
                                  gost28147 chacha20 sm4 aria
        Enabled digest algorithms: crc gostr3411-94 md4 md5 rmd160 sha1
                                  sha256 sha512 sha3 tiger whirlpool stribog
                                  blake2 sm3
        Enabled kdf algorithms:   s2k pkdf2 scrypt
        Enabled pubkey algorithms: dsa elgamal rsa ecc
        Random number generator:  default
        Try using jitter entropy: yes
        Using linux capabilities: no
        FIPS module version:
        Try using Padlock crypto: yes
        Try using AES-NI crypto:  yes
        Try using Intel SHAEXT:   yes
        Try using Intel PCLMUL:   yes
        Try using Intel SSE4.1:   yes
        Try using DRNG (RDRAND):  yes
        Try using Intel AVX:      yes
        Try using Intel AVX2:     yes
        Try using Intel AVX512:   yes
        Try using Intel GFNI:     yes
        Try using ARM NEON:       n/a
        Try using ARMv8 crypto:   n/a
        Try using ARMv8 SVE:      n/a
        Try using ARMv9 SVE2:     n/a
        Try using PPC crypto:     n/a

root@debian:/usr/local/src/libgcrypt-1.11.0#
```

Figure 11 : Installing libgcrypt continued

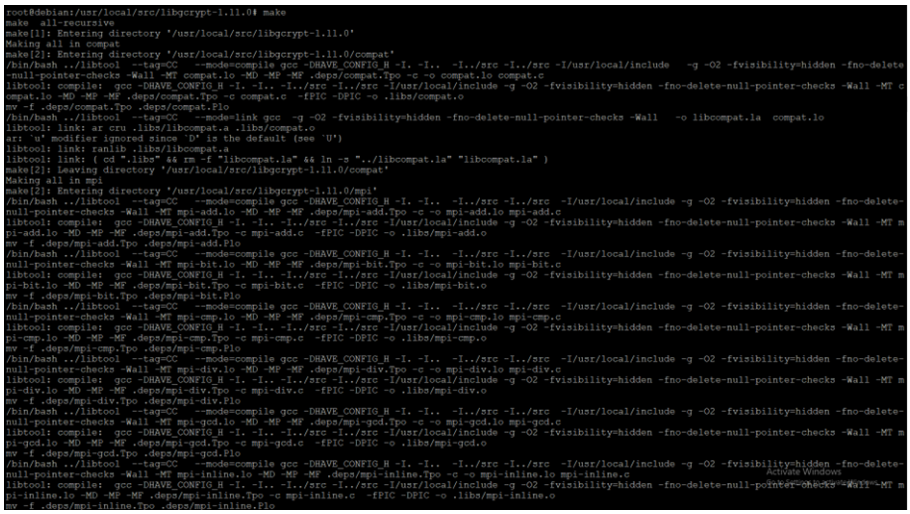

Figure 12 : Installing libgcrypt_make

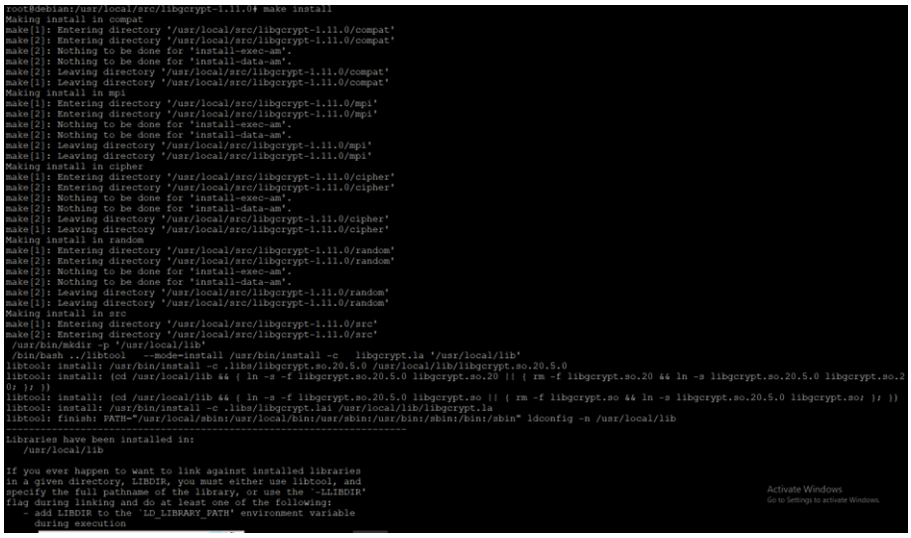

Figure 13 : Installing libgcrypt_make install

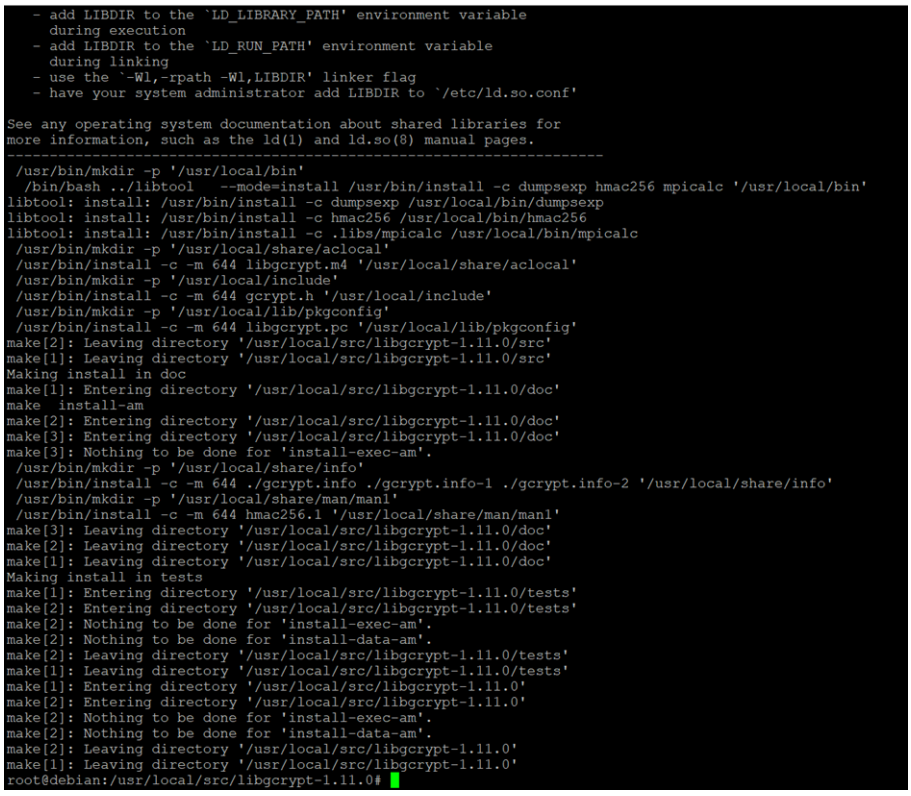

Figure 14 : Installing libgcrypt_make install continued

5.1.3 Installing Libassuan

1. Download the `libassuan` installation file.

```
# wget https://www.gnupg.org/ftp/gcrypt/libassuan/libassuan-3.0.2.tar.bz2
```

Figure 15 : Downloading libgassuan

2. Extract the file.

```
# tar -xjf libassuan-3.0.2.tar.bz2
```

3. Go to the directory where the file is extracted.

```
# cd libassuan-3.0.2
```

4. Run the following command to compile and install.

```
# ./configure --prefix=/usr/local
# make
# make install
```

```
root@debian:/usr/local/src/libassuan-3.0.2# ./configure --prefix=/usr/local
checking for a BSD-compatible install... /usr/bin/install -c
checking whether build environment is sane... yes
checking for a race-free mkdir -p... /usr/bin/mkdir -p
checking for gawk... no
checking for mawk... mawk
checking whether make sets $(MAKE)... yes
checking whether make supports nested variables... yes
checking whether to enable maintainer-specific portions of Makefiles... no
checking build system type... x86_64-pc-linux-gnu
checking host system type... x86_64-pc-linux-gnu
checking whether make supports nested variables... (cached) yes
configure: autobuild project... libassuan
configure: autobuild revision... 3.0.2
configure: autobuild hostname... debian
configure: autobuild timestamp... 20260406-040949
checking whether make supports the include directive... yes (GNU style)
checking for gcc... gcc
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 the compiler supports GNU C... yes
checking whether gcc accepts -g... yes
checking for gcc option to enable C11 features... none needed
checking whether gcc understands -c and -o together... yes
checking dependency style of gcc... gcc3
checking for stdio.h... yes
checking for stdlib.h... yes
checking for string.h... yes
checking for inttypes.h... yes
checking for stdint.h... yes
checking for strings.h... yes
checking for sys/stat.h... yes
checking for sys/types.h... yes
checking for unistd.h... yes
checking for wchar.h... yes
checking for minix/config.h... no
checking for sys/socket.h... yes
checking whether it is safe to define __EXTENSIONS__... yes
checking whether _XOPEN_SOURCE should be defined... no
checking how to print strings... printf
checking for a sed that does not truncate output... /usr/bin/sed
checking for grep that handles long lines and -e... /usr/bin/grep
checking for egrep... /usr/bin/grep -E
checking for fgrep... /usr/bin/grep -F
checking for ld used by gcc... /usr/bin/ld
```

Figure 16 : Installing libassuan

```
checking for sys/ucred.h... no
checking for uintptr_t... yes
checking for uint16_t... yes
checking for an ANSI C-conforming const... yes
checking for inline... inline
checking for size_t... yes
checking for socklen_t... yes
checking for struct cmsghdr.cmsg_len... yes
checking for gpg-error-config... no
checking for gpgrt-config... /usr/local/bin/gpgrt-config
configure: Use gpgrt-config with /usr/local/lib as gpg-error-config
checking for GPG Error - version >= 1.17... yes (1.59)
checking for flockfile... yes
checking for funlockfile... yes
checking for inet_pton... yes
checking for stat... yes
checking for getaddrinfo... yes
checking for getrlimit... yes
checking for library containing nanosleep... none required
checking for funopen... no
checking for fopencookie... yes
checking for isascii... yes
checking for memrchr... yes
checking for stpcpy... yes
checking for unistd.h... (cached) yes
checking for setenv... yes
checking for struct sockpeercred.pid... no
checking for getpeerucred... no
checking for getpeereid... no
checking for struct xucred.cr_pid... no
checking that generated files are newer than configure... done
configure: creating ./config.status
config.status: creating Makefile
config.status: creating m4/Makefile
config.status: creating src/Makefile
config.status: creating doc/Makefile
config.status: creating tests/Makefile
config.status: creating src/libassuan-config
config.status: creating src/versioninfo.rc
config.status: creating src/libassuan.pc
config.status: creating config.h
config.status: executing depfiles commands
config.status: executing libtool commands

        Libassuan v3.0.2 has been configured as follows:

        Revision: 0f84595  (3972)
        Platform: x86_64-pc-linux-gnu

root@debian:/usr/local/src/libassuan-3.0.2#
```

Figure 17 : Installing libassuan continued

Figure 18 : Installing libassuan_make

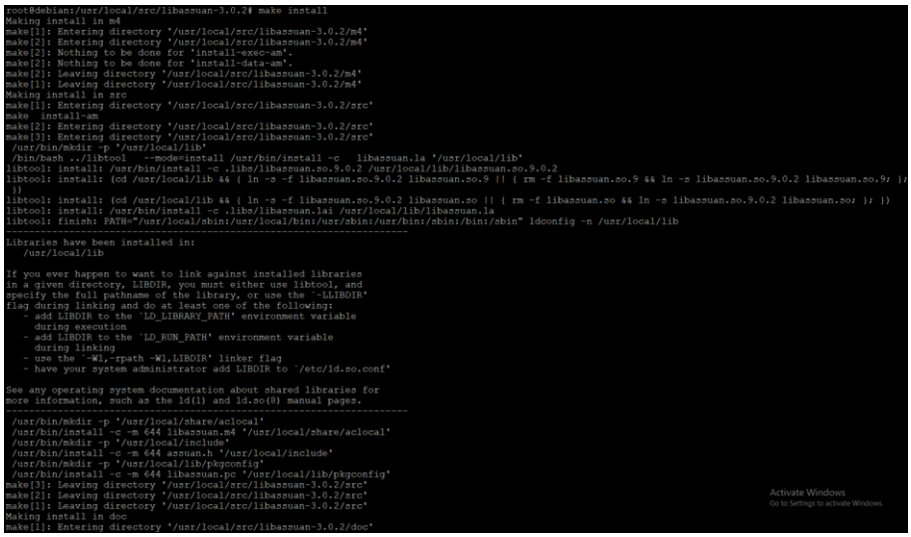

Figure 19 : Installing libassuan_make install

5.1.4 Installing Libksba

1. Download the `libksba` installation file.

```
# wget https://www.gnupg.org/ftp/gcrypt/libksba/libksba-1.6.6.tar.bz2
```

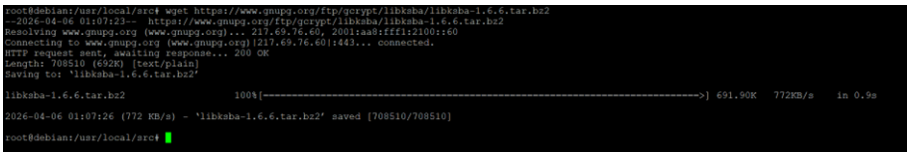

Figure 20 : Downloading libksba

2. Extract the file.

```
# tar -xjf libksba-1.6.6.tar.bz2
```

3. Go to the directory where the file is extracted.

```
# cd libksba-1.6.6
```

4. Run the following command to compile and install.

```
# ./configure --prefix=/usr/local
# make
# make install
```

```
root@debian:/usr/local/src/libksba-1.6.6# ./configure --prefix=/usr/local
checking for a BSD-compatible install... /usr/bin/install -c
checking whether build environment is sane... yes
checking for a race-free mkdir -p... /usr/bin/mkdir -p
checking for gawk... no
checking for mawk... mawk
checking whether make sets $(MAKE)... yes
checking whether make supports nested variables... yes
checking build system type... x86_64-pc-linux-gnu
checking host system type... x86_64-pc-linux-gnu
configure: autobuild project... libksba
configure: autobuild revision... 1.6.6
configure: autobuild hostname... debian
configure: autobuild timestamp... 20260406-010802
checking whether make supports the include directive... yes (GNU style)
checking for gcc... gcc
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 the compiler supports GNU C... yes
checking whether gcc accepts -g... yes
checking for gcc option to enable C11 features... none needed
checking whether gcc understands -c and -o together... yes
checking dependency style of gcc... gcc3
checking for stdio.h... yes
checking for stdlib.h... yes
checking for string.h... yes
checking for inttypes.h... yes
checking for stdint.h... yes
checking for strings.h... yes
checking for sys/stat.h... yes
checking for sys/types.h... yes
checking for unistd.h... yes
checking for wchar.h... yes
checking for minix/config.h... no
checking whether it is safe to define __EXTENSIONS__... yes
checking whether _XOPEN_SOURCE should be defined... no
checking how to print strings... printf
checking for a sed that does not truncate output... /usr/bin/sed
checking for grep that handles long lines and -e... /usr/bin/grep
checking for egrep... /usr/bin/grep -E
checking for fgrep... /usr/bin/grep -F
checking for ld used by gcc... /usr/bin/ld
```

Figure 21 : Installing libksba

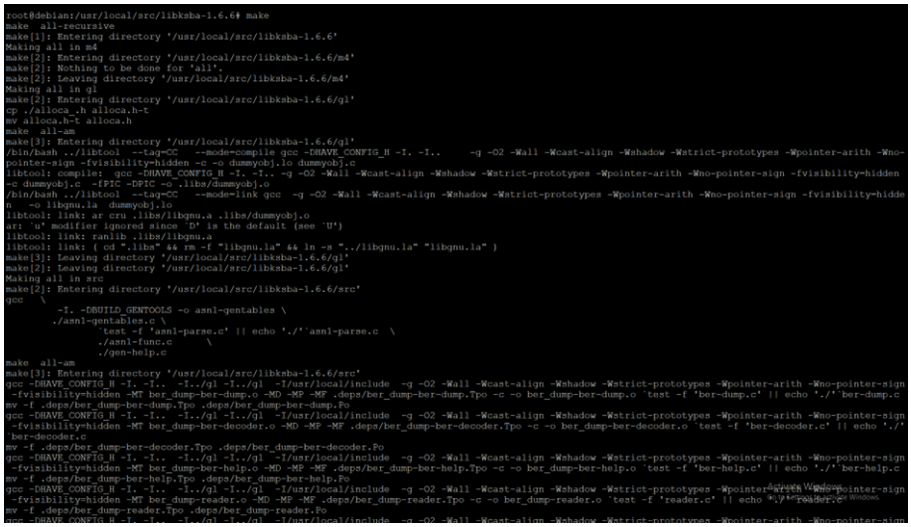

Figure 22 : Installing libksba_make

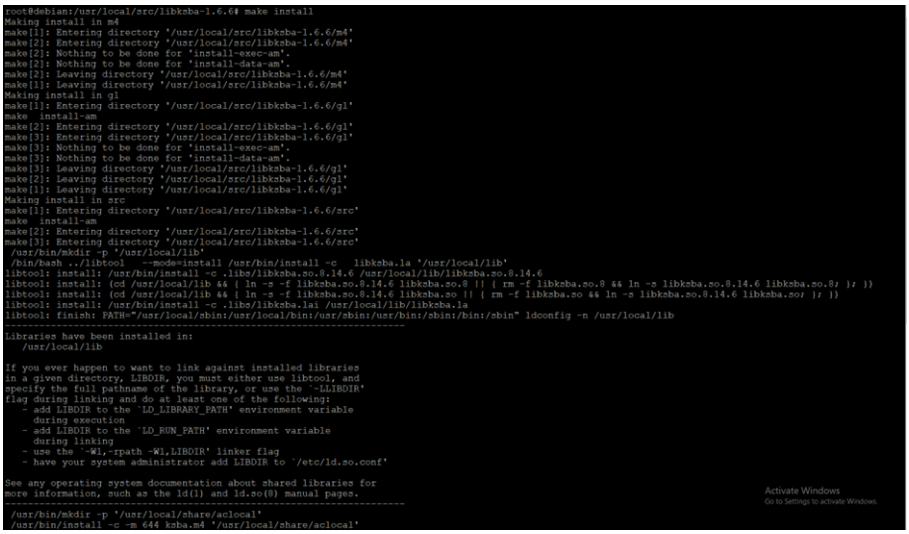

Figure 23 : Installing libksba_make install

5.1.5 Installing NPTH

1. Download the `npth` installation file.

```
# wget https://www.gnupg.org/ftp/gcrypt/npth/npth-1.8.tar.bz2
```

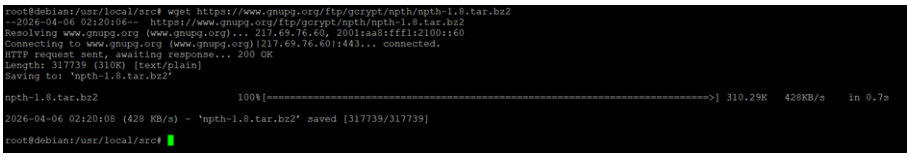

Figure 24 : Downloading npth

2. Extract the file.

```
# tar -xjf npth-1.8.tar.bz2
```

3. Go to the directory where the file is extracted.

```
# cd npth-1.8
```

4. Run the following command to compile and install.

```
# ./configure --prefix=/usr/local
# make
# make install
```

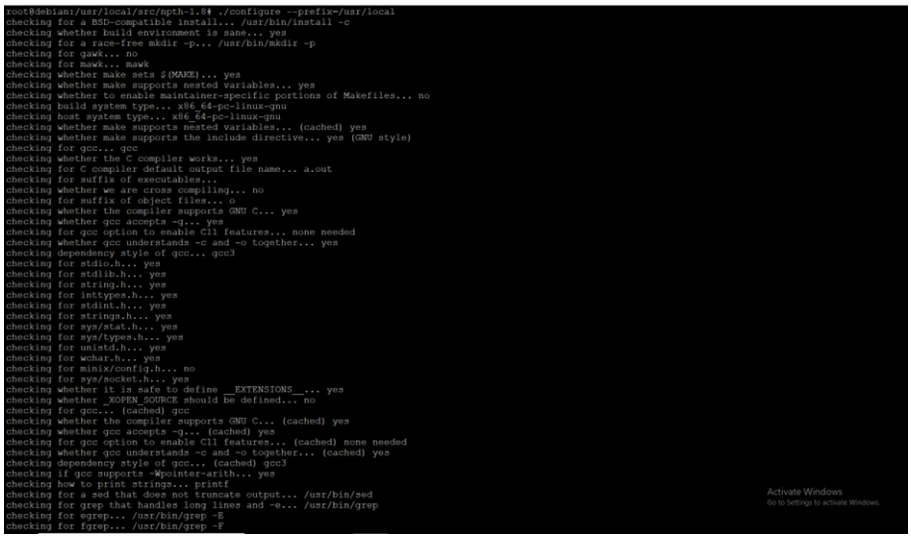

Figure 25 : Installing npth

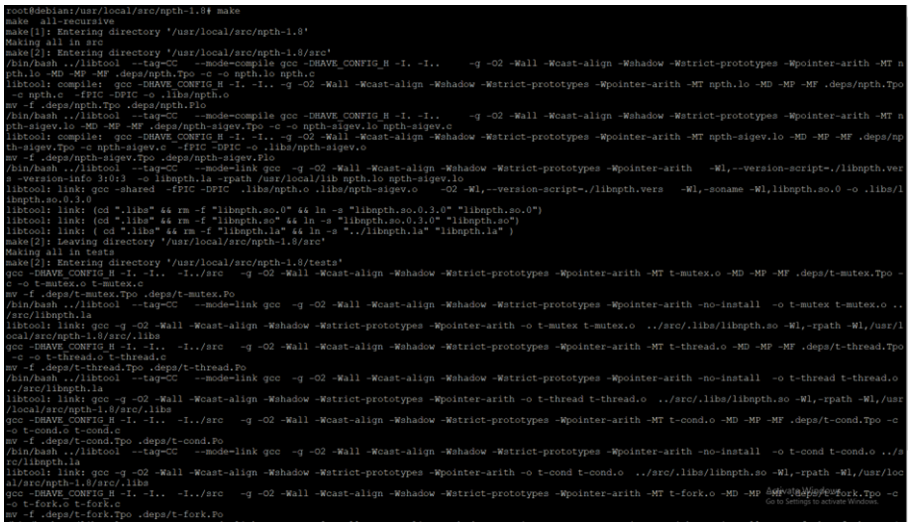

Figure 26 : Installing npth_make

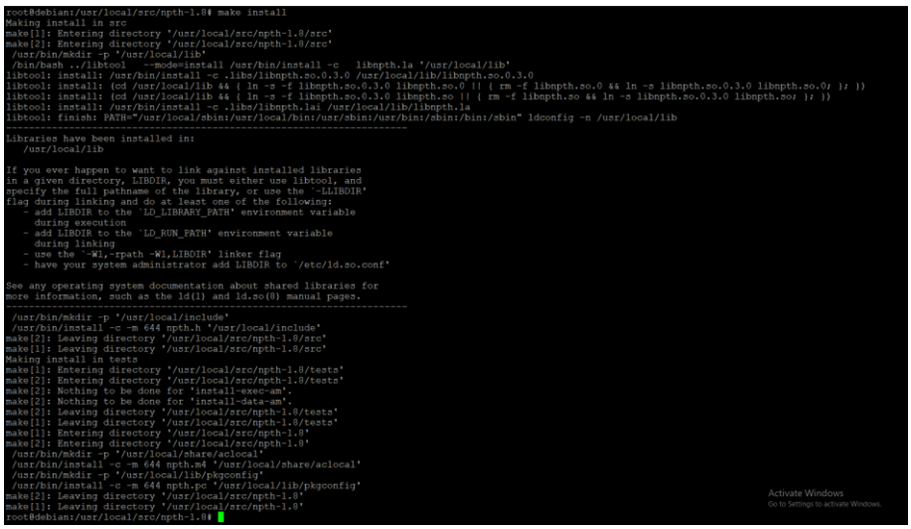

Figure 27 : Installing npth_make install

Figure 28 : Installing npth_verification

5.1.6 Installing Pinentry

1. Download the `pinentry` installation file.

```
# wget https://www.gnupg.org/ftp/gcrypt/pinentry/pinentry-1.3.1.tar.bz2
```

Figure 29 : Downloading pinentry

2. Extract the file.

```
# tar -xjf pinentry-1.3.1.tar.bz2
```

3. Go to the directory where the file is extracted.

```
# cd pinentry-1.3.1
```

4. Run the following command to compile and install.

```
# ./configure --prefix=/usr/local --enable-pinentry-curses
# make
# make install
```

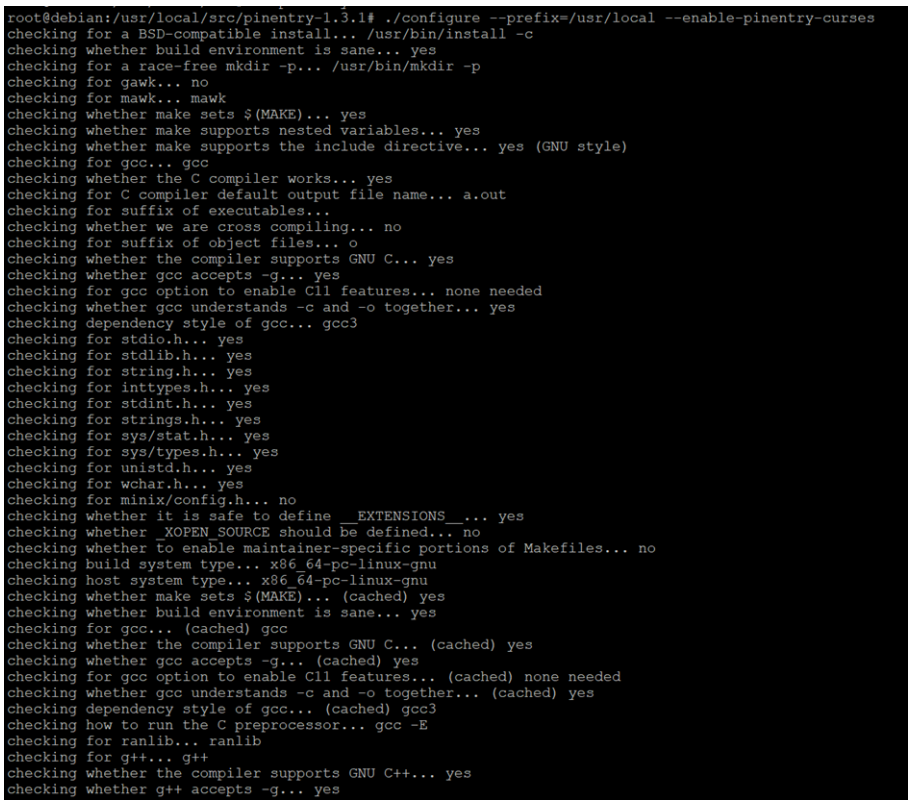

Figure 30 : Installing pinentry

```
configure:

        Pinentry v1.3.1 has been configured as follows:

        Revision:  dd8894f  (56712)
        Platform:  x86_64-pc-linux-gnu

        Curses Pinentry ..: yes
        TTY Pinentry .....: maybe
        Emacs Pinentry ...: no
        EFL Pinentry .....: no
        GTK+-2 Pinentry ..: no
        GNOME 3 Pinentry .: no
        Qt6 Pinentry .....: no
        Qt5 Pinentry .....: no
        Qt4 Pinentry .....: no
        TQt Pinentry .....: no
        W32 Pinentry .....: no
        FLTK Pinentry ....: no

        Fallback to Curses: yes
        Emacs integration : yes

        libsecret ........: no

        Default Pinentry .: pinentry-curses

root@debian:/usr/local/src/pinentry-1.3.1#
```

Figure 31 : Installing pinentry_continued

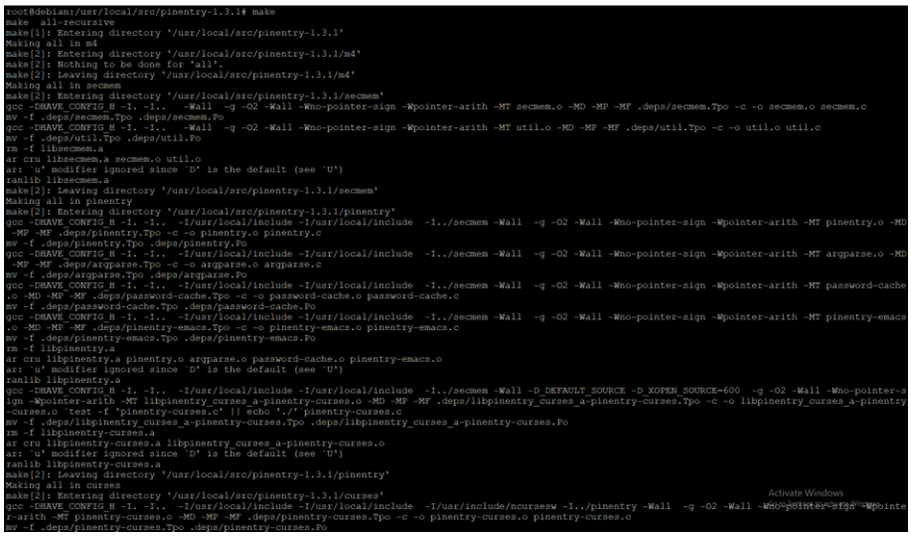

Figure 32 : Installing pinentry_make

```
root@debian:/usr/local/src/pinentry-1.3.1# make install
Making install in m4
make[1]: Entering directory '/usr/local/src/pinentry-1.3.1/m4'
make[2]: Entering directory '/usr/local/src/pinentry-1.3.1/m4'
make[2]: Nothing to be done for 'install-exec-am'.
make[2]: Nothing to be done for 'install-data-am'.
make[2]: Leaving directory '/usr/local/src/pinentry-1.3.1/m4'
make[1]: Leaving directory '/usr/local/src/pinentry-1.3.1/m4'
Making install in secmem
make[1]: Entering directory '/usr/local/src/pinentry-1.3.1/secmem'
make[2]: Entering directory '/usr/local/src/pinentry-1.3.1/secmem'
make[2]: Nothing to be done for 'install-exec-am'.
make[2]: Nothing to be done for 'install-data-am'.
make[2]: Leaving directory '/usr/local/src/pinentry-1.3.1/secmem'
make[1]: Leaving directory '/usr/local/src/pinentry-1.3.1/secmem'
Making install in pinentry
make[1]: Entering directory '/usr/local/src/pinentry-1.3.1/pinentry'
make[2]: Entering directory '/usr/local/src/pinentry-1.3.1/pinentry'
make[2]: Nothing to be done for 'install-exec-am'.
make[2]: Nothing to be done for 'install-data-am'.
make[2]: Leaving directory '/usr/local/src/pinentry-1.3.1/pinentry'
make[1]: Leaving directory '/usr/local/src/pinentry-1.3.1/pinentry'
Making install in curses
make[1]: Entering directory '/usr/local/src/pinentry-1.3.1/curses'
make[2]: Entering directory '/usr/local/src/pinentry-1.3.1/curses'
 /usr/bin/mkdir -p '/usr/local/bin'
  /usr/bin/install -c pinentry-curses '/usr/local/bin'
make[2]: Nothing to be done for 'install-data-am'.
make[2]: Leaving directory '/usr/local/src/pinentry-1.3.1/curses'
make[1]: Leaving directory '/usr/local/src/pinentry-1.3.1/curses'
Making install in doc
make[1]: Entering directory '/usr/local/src/pinentry-1.3.1/doc'
make[2]: Entering directory '/usr/local/src/pinentry-1.3.1/doc'
make[2]: Nothing to be done for 'install-exec-am'.
 /usr/bin/mkdir -p '/usr/local/share/info'
 /usr/bin/install -c -m 644 ./pinentry.info '/usr/local/share/info'
make[2]: Leaving directory '/usr/local/src/pinentry-1.3.1/doc'
make[1]: Leaving directory '/usr/local/src/pinentry-1.3.1/doc'
make[1]: Entering directory '/usr/local/src/pinentry-1.3.1'
make[2]: Entering directory '/usr/local/src/pinentry-1.3.1'
(cd /usr/local/bin; \
rm -f pinentry; \
ln -s pinentry-curses pinentry)
make[2]: Nothing to be done for 'install-data-am'.
make[2]: Leaving directory '/usr/local/src/pinentry-1.3.1'
make[1]: Leaving directory '/usr/local/src/pinentry-1.3.1'
root@debian:/usr/local/src/pinentry-1.3.1#
```

Figure 33 : Installing pinentry_make install

```
root@debian:/usr/local/src/pinentry-1.3.1# /usr/local/bin/pinentry --version
pinentry-curses (pinentry) 1.3.1
Copyright (C) 2016 g10 Code GmbH
License GPLv2+: GNU GPL version 2 or later <https://www.gnu.org/licenses/>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
root@debian:/usr/local/src/pinentry-1.3.1#
```

Figure 34 : Installing pinentry_version

5.2 Installing GnuPG

1. Download the GnuPG installation file.

```
# wget https://www.gnupg.org/ftp/gcrypt/gnupg/gnupg-2.5.18.tar.bz2
```

Figure 35 : Downloading gnupg

2. Extract the file.

```
# tar -xjf gnupg-2.5.18.tar.bz2
```

3. Go to the directory where the file is extracted.

```
# cd gnupg-2.5.18
```

4. Run the following command to compile and install.

```
# ./configure --prefix=/usr/local
# make
# make install
```

```
root@debian:/usr/local/src# cd gnupg-2.5.18
root@debian:/usr/local/src/gnupg-2.5.18# ./configure --prefix=/usr/local
checking for a BSD-compatible install... /usr/bin/install -c
checking whether build environment is sane... yes
checking for a thread-safe mkdir -p... /usr/bin/mkdir -p
checking for gawk... no
checking for mawk... mawk
checking whether make sets $(MAKE)... yes
checking whether make supports nested variables... yes
checking build system type... x86_64-pc-linux-gnu
checking host system type... x86_64-pc-linux-gnu
configure: autobuild project... gnupg
configure: autobuild revision... 2.5.18
configure: autobuild hostname... debian
configure: autobuild timestamp... 20260406-041916
checking whether make supports the include directive... yes (GNU style)
checking for gcc... gcc
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 accepts -g... yes
checking for gcc option to accept ISO C89... none needed
checking whether gcc understands -c and -o together... yes
checking dependency style of gcc... gcc3
checking how to run the C preprocessor... gcc -E
checking for grep that handles long lines and -e... /usr/bin/grep
checking for egrep... /usr/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
checking for string.h... yes
checking for memory.h... yes
checking for strings.h... yes
checking for inttypes.h... yes
checking for stdint.h... yes
checking for unistd.h... yes
checking minix/config.h usability... no
checking minix/config.h presence... no
checking for minix/config.h... no
checking whether it is safe to define __EXTENSIONS__... yes
checking whether SELinux support is requested... no
checking whether to allocate extra secure memory... no
checking calibrated passphrase-stretching (s2k) duration... 100 milliseconds
checking whether to enable trust models... yes
checking whether to enable TOFU... yes
```

Figure 36 : Installing gpg

```
        GnuPG v2.5.18 has been configured as follows:

        Revision:  1b8362889  (7043)
        Platform:  GNU/Linux (x86_64-pc-linux-gnu)

        OpenPGP:   yes
        S/MIME:    yes
        Agent:     yes
        Smartcard: yes (without internal CCID driver)
        TPM:       no
        G13:       no
        Dirmngr:   no
        Keyboxd:   no
        Gpgtar:    yes
        WKS tools: yes

        Protect tool:       (default)
        LDAP wrapper:       (default)
        Default agent:      (default)
        Default pinentry:   (default)
        Default scdaemon:   (default)
        Default keyboxd:    (default)
        Default tpm2daemon: (default)
        Default dirmngr:    (default)

        Dirmngr auto start:  yes
        Readline support:    no
        LDAP support:        n/a
        TLS support:         no
        TOFU support:        no
        Tor support:         only .onion

root@debian:/usr/local/src/gnupg-2.5.18#
```

Figure 37 : Installing gpg_continued

```
root@debian:/usr/local/src/gnupg-2.5.18# make install
Making install in m4
make[1]: Entering directory '/usr/local/src/gnupg-2.5.18/m4'
make[2]: Entering directory '/usr/local/src/gnupg-2.5.18/m4'
make[2]: Nothing to be done for 'install-exec-am'.
make[2]: Nothing to be done for 'install-data-am'.
make[2]: Leaving directory '/usr/local/src/gnupg-2.5.18/m4'
make[1]: Leaving directory '/usr/local/src/gnupg-2.5.18/m4'
Making install in common
make[1]: Entering directory '/usr/local/src/gnupg-2.5.18/common'
make  install-am
make[2]: Entering directory '/usr/local/src/gnupg-2.5.18/common'
make[3]: Entering directory '/usr/local/src/gnupg-2.5.18/common'
make[3]: Nothing to be done for 'install-exec-am'.
make[3]: Nothing to be done for 'install-data-am'.
make[3]: Leaving directory '/usr/local/src/gnupg-2.5.18/common'
make[2]: Leaving directory '/usr/local/src/gnupg-2.5.18/common'
make[1]: Leaving directory '/usr/local/src/gnupg-2.5.18/common'
Making install in regexp
make[1]: Entering directory '/usr/local/src/gnupg-2.5.18/regexp'
make  install-am
make[2]: Entering directory '/usr/local/src/gnupg-2.5.18/regexp'
make[3]: Entering directory '/usr/local/src/gnupg-2.5.18/regexp'
make[3]: Nothing to be done for 'install-exec-am'.
make[3]: Nothing to be done for 'install-data-am'.
make[3]: Leaving directory '/usr/local/src/gnupg-2.5.18/regexp'
make[2]: Leaving directory '/usr/local/src/gnupg-2.5.18/regexp'
make[1]: Leaving directory '/usr/local/src/gnupg-2.5.18/regexp'
Making install in kbx
make[1]: Entering directory '/usr/local/src/gnupg-2.5.18/kbx'
make[2]: Entering directory '/usr/local/src/gnupg-2.5.18/kbx'
 /usr/bin/mkdir -p '/usr/local/bin'
  /usr/bin/install -c kbxutil '/usr/local/bin'
make[2]: Nothing to be done for 'install-data-am'.
make[2]: Leaving directory '/usr/local/src/gnupg-2.5.18/kbx'
make[1]: Leaving directory '/usr/local/src/gnupg-2.5.18/kbx'
Making install in g10
make[1]: Entering directory '/usr/local/src/gnupg-2.5.18/g10'
make[2]: Entering directory '/usr/local/src/gnupg-2.5.18/g10'
 /usr/bin/mkdir -p '/usr/local/bin'
  /usr/bin/install -c gpg gpgv '/usr/local/bin'
/bin/bash ../build-aux/mkinstalldirs /usr/local/share/gnupg
mkdir -p -- /usr/local/share/gnupg
/usr/bin/install -c -m 644 ./distsigkey.gpg \
                     /usr/local/share/gnupg/distsigkey.gpg
make[2]: Leaving directory '/usr/local/src/gnupg-2.5.18/g10'
make[1]: Leaving directory '/usr/local/src/gnupg-2.5.18/g10'
Making install in sm
make[1]: Entering directory '/usr/local/src/gnupg-2.5.18/sm'
make[2]: Entering directory '/usr/local/src/gnupg-2.5.18/sm'
```

Figure 38 : Installing gpg_make install

```
make[2]: Leaving directory '/usr/local/src/gnupg-2.5.18/tests/gpgme'
Making install in pkits
make[2]: Entering directory '/usr/local/src/gnupg-2.5.18/tests/pkits
make[3]: Entering directory '/usr/local/src/gnupg-2.5.18/tests/pkits
make[3]: Nothing to be done for 'install-exec-am'.
make[3]: Nothing to be done for 'install-data-am'.
make[3]: Leaving directory '/usr/local/src/gnupg-2.5.18/tests/pkits'
make[2]: Leaving directory '/usr/local/src/gnupg-2.5.18/tests/pkits'
Making install in .
make[2]: Entering directory '/usr/local/src/gnupg-2.5.18/tests'
make[3]: Entering directory '/usr/local/src/gnupg-2.5.18/tests'
make[3]: Nothing to be done for 'install-exec-am'.
make[3]: Nothing to be done for 'install-data-am'.
make[3]: Leaving directory '/usr/local/src/gnupg-2.5.18/tests'
make[2]: Leaving directory '/usr/local/src/gnupg-2.5.18/tests'
make[1]: Leaving directory '/usr/local/src/gnupg-2.5.18/tests'
make[1]: Entering directory '/usr/local/src/gnupg-2.5.18'
(set -e; cd bin; \
 for i in gpg gpgv; \
     do ln -sf ../g10/$i .; done; \
 for i in gpgsm; \
     do ln -sf ../sm/$i .; done; \
 for i in gpg-agent; \
     do ln -sf ../agent/$i .; done; \
 for i in dirmngr; \
     do ln -sf ../dirmngr/$i .; done; \
 for i in gpgconf gpg-connect-agent gpgtar gpg-card; \
     do ln -sf ../tools/$i .; done; \
 cd ../libexec ; \
 for i in keyboxd; \
     do ln -sf ../kbx/$i .; done; \
 for i in scdaemon; \
     do ln -sf ../scd/$i .; done; \
 for i in gpg-preset-passphrase; \
     do ln -sf ../agent/$i .; done; \
 for i in tpm2daemon; \
            do [ -f ../tpm2d/$i ] && ln -sf ../tpm2d/$i .; done; \
 echo "created links to binaries" )
created links to binaries
make[2]: Entering directory '/usr/local/src/gnupg-2.5.18'
make[2]: Nothing to be done for 'install-exec-am'.
 /usr/bin/mkdir -p '/usr/local/share/doc/gnupg'
 /usr/bin/install -c -m 644 README '/usr/local/share/doc/gnupg'
make  install-data-hook
make[3]: Entering directory '/usr/local/src/gnupg-2.5.18'
make[3]: Nothing to be done for 'install-data-hook'.
make[3]: Leaving directory '/usr/local/src/gnupg-2.5.18'
make[2]: Leaving directory '/usr/local/src/gnupg-2.5.18'
make[1]: Leaving directory '/usr/local/src/gnupg-2.5.18'
root@debian:/usr/local/src/gnupg-2.5.18#
```

Figure 39 : Installing gpg_make install continued

5. Export the `PATH` and `LD_LIBRARY_PATH` variables.

```
# export PATH=/usr/local/bin:$PATH
# export LD_LIBRARY_PATH=/usr/local/lib
```

6. Verify that the GnuPG has been installed successfully.

```
# gpg --version
```

Figure 40 : Verifying gpg version

5.3 Installing GnuPG-PKCS11-SCD

1. Install the `pkcs11-helper-devel` package.

```
# sudo apt update
# sudo apt install -y libssl-dev
# pkg-config --modversion libcrypto
# sudo apt update
# sudo apt install -y libpkcs11-helper1-dev
```

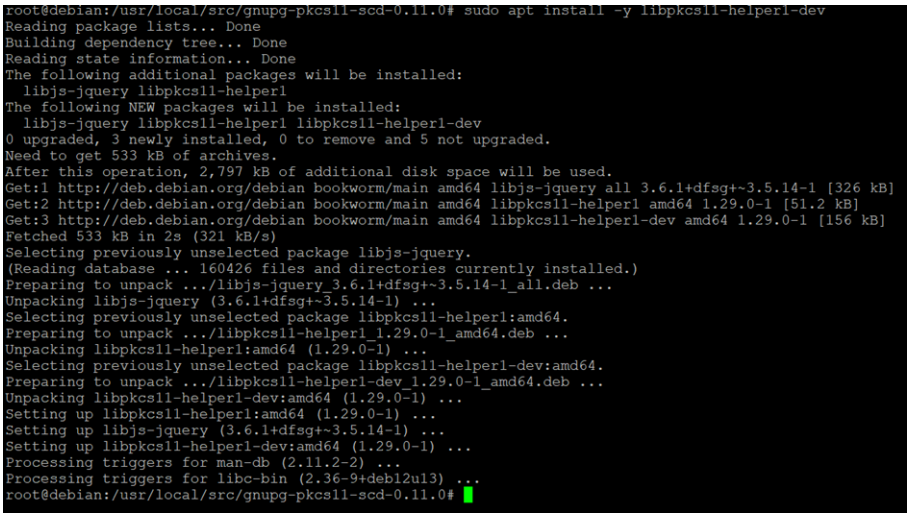

Figure 41 : Installation of pkcs11-helper-devel package

2. Verify that the PKCS#11 helper library is installed and available for development.

```
# dpkg -L libpkcs11-helper1-dev | grep pkgconfig
# export PKG_CONFIG_PATH=/usr/lib/x86_64-linux-gnu/pkgconfig
# pkg-config --modversion libpkcs11-helper-1
```

3. Download the `gnupg-pkcs11-scd` installation file.

```
# wget https://github.com/alonbl/gnupg-pkcs11-scd/releases/download/gnupg-pkcs11-
scd-0.11.0/gnupg-pkcs11-scd-0.11.0.tar.bz2
```

Figure 42 : Downloading gnupg-pkcs11-scd

4. Extract the file.

```
# tar -xjf gnupg-pkcs11-scd-0.11.0.tar.bz2
```

5. Go to the directory where the file is extracted.

```
# cd gnupg-pkcs11-scd-0.11.0
```

6. Run the following command to compile and install.

```
# ./configure
# make
# make install
```

Figure 43 : Installation of pkcs11-helper-devel package_make

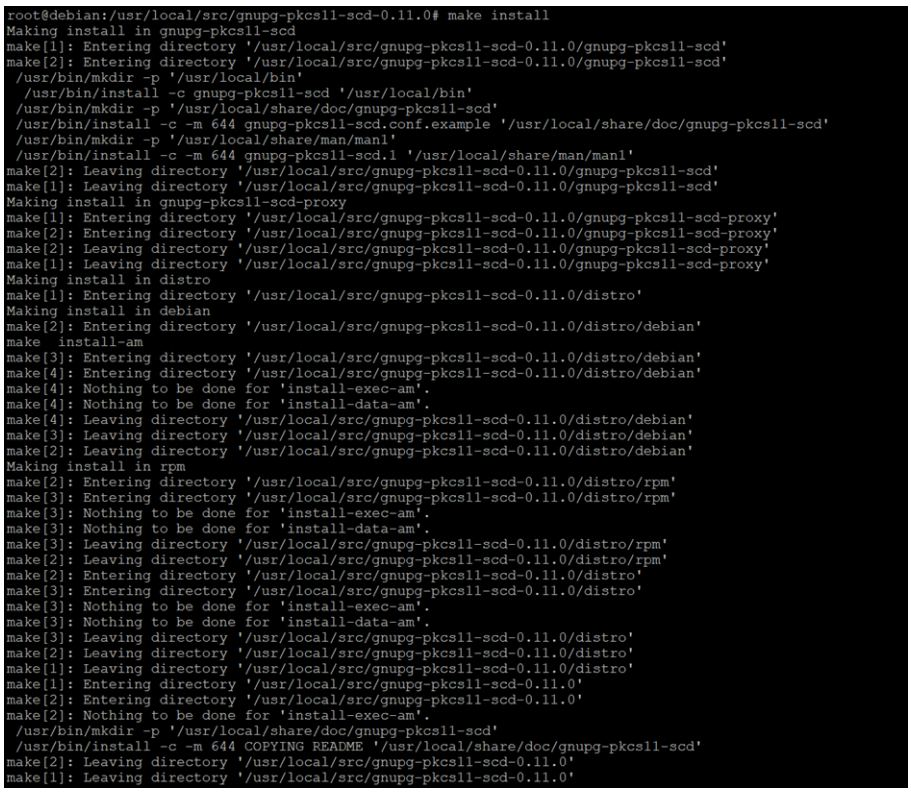

Figure 44 : Installation of pkcs11-helper-devel package_make install

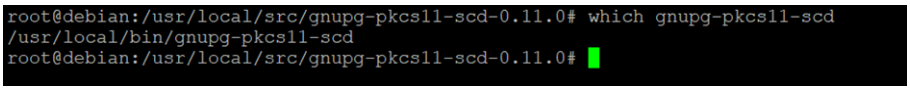

Figure 45 : Installing gnupg-pkcs11-scd_verification

6 Configuring GnuPG to Use Utimaco HSM

6.1 Setting up Utimaco CryptoServer library in gnupg-pkcs11-scd Configuration File

1. Run the following command to automatically create the directory structure for GnuPG.

```
# gpg --list-keys
```

```
root@debian:~# gpg --list-keys
root@debian:~# gpg --list-keys --verbose
gpg: enabled compatibility flags:
gpg: using pgp trust model
root@debian:~# ls -ld /root/.gnupg
drwx------ 3 root root 4096 Apr  7 06:07 /root/.gnupg
```

Figure 46 : Listing gpg keys

2. Copy the sample file from `resource/gnupg-pkcs11-scd-0.10.0/gnupg-pkcs11-scd/gnupg-pkcs11-scd.conf.example` to `~/.gnupg/gnupg-pkcs11-scd.conf`.

```
# cp /usr/local/src/gnupg-pkcs11-scd-0.11.0/gnupg-pkcs11-scd/gnupg-pkcs11-
scd.conf.example /root/.gnupg/gnupg-pkcs11-scd.conf
```

3. Open the file `/root/.gnupg/gnupg-pkcs11-scd.conf` and make the following changes.

```
# pin-cache 0
# providers p1
# provider-p1-library /etc/utimaco/lib/libcs_pkcs11_R3.so
```

4. Create a file `/root/.gnupg/gpg-agent.conf` and add the following content in it.

```
# scdaemon-program /usr/local/bin/gnupg-pkcs11-scd
# pinentry-program /usr/local/bin/pinentry
```

6.2 Generating Key and Certificate for GnuPG

1. Generate the key pair using the below `p11tool2` command.

```
# ./p11tool2 Slot=0 LoginUser=Gnupg1234
PubKeyAttr=CKA_LABEL="GPGPublicKey",CKA_MODULUS_BITS=2048,CKA_ID=0x45
PrvKeyAttr=CKA_LABEL="GPGPrivateKey",CKA_EXTRACTABLE=CK_TRUE,CKA_ID=0x45
GenerateKeyPair=RSA
```

Only RSA keys are supported with GnuPG PKCS11 SCD.

2. Verify that the keys are generated.

```
# ./p11tool2 Slot=0 LoginUser=Gnupg1234 ListObjects
```

Figure 47 : Listing keys on HSM slot

3. Install `opensc` and `openssl-pkcs11`.

```
# apt update
# apt install -y libengine-pkcs11-openssl
```

4. Open `openssl` shell and load the dynamic engine.

```
# openssl engine dynamic \
  -pre SO_PATH:/opt/openssl-1.1.1/lib/engines-1.1/pkcs11.so \
  -pre ID:pkcs11 \
  -pre LIST_ADD:1 \
  -pre LOAD \
  -pre MODULE_PATH:/etc/utimaco/lib/libcs_pkcs11_R3.so
```

Figure 48 : Loading dynamic engine in openssl

5. Run the following command to generate a self-signed certificate. Provide slot PIN when prompted.

```
openssl req -new -x509
  -engine pkcs11 \
  -keyform engine \
  -key "pkcs11:id=%45;type=private" \
  -sha256 \
  -days 365 \
  -out test.pem \
  -subj "/CN=test.utimaco.com/O=Integration"
```

Figure 49 : Generating self-signed certificate

After this, a certificate `test.pem` is generated.

Type “exit” to exit from `openssl` prompt.

It is recommended to use a CA signed certificate for production environment.

6. Convert the certificate from `pem` to `der`.

```
# openssl x509 \
  -in test.pem \
  -out test.der \
  -outform DER
```

7. Import the certificate to Utimaco HSM.

```
# pkcs11-tool \
  --module /etc/utimaco/lib/libcs_pkcs11_R3.so \
  -l \
  --slot 0 \
  --write-object test.der \
  --type cert \
  --id 45 \
  --label "GPGCertificate"
```

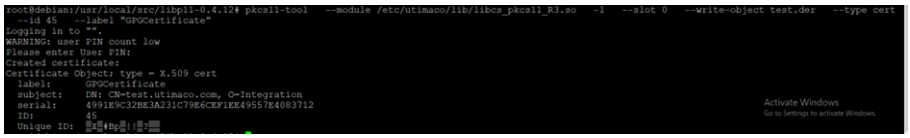

Figure 50 : Importing certificate to Utimaco HSM

8. Verify that the certificate has been imported to Utimaco HSM.

```
# p11tool2 slot=0 LoginUser=Gnupg1234 ListObjects
```

```
root@debian:/etc/utimaco/bin# ./p11tool2 Slot=0 LoginUser=Gnupg1234 ListObjects

CKO_CERTIFICATE:

+ 1.1
  CKA_CERTIFICATE_TYPE          = CKC_X_509
  CKA_UNIQUE_ID                 = 9ECFAAE5-B523-4270-A57C-7C943FEFFC97
  CKA_LABEL                     = GPGCertificate
  CKA_ID                        = 0x45 (E)
  CKA_SUBJECT                   =
                0x30313119 30170603 5504030C 10746573 |011 0   U    tes|
                  742E7574 696D6163 6F2E636F 6D311430 |t.utimaco.com1 0|
                  12060355 040A0C0B 496E7465 67726174 |   U    Integrat|
                  696F6E                              |ion             |


CKO_PUBLIC_KEY:

+ 2.1
  CKA_KEY_TYPE                  = CKK_RSA
  CKA_UNIQUE_ID                 = 89EC0950-BE3B-4AA4-8460-A9FC2009F650
  CKA_LABEL                     = GPGPublicKey
  CKA_ID                        = 0x45 (E)


CKO_PRIVATE_KEY:

+ 3.1
  CKA_KEY_TYPE                  = CKK_RSA
  CKA_UNIQUE_ID                 = 8F6FDA38-E5C8-4121-84EF-CD5383D40708
  CKA_SENSITIVE                 = CK_TRUE
  CKA_EXTRACTABLE               = CK_TRUE
  CKA_LABEL                     = GPGPrivateKey
  CKA_ID                        = 0x45 (E)
```

Figure 51 : Listing keys and certificate on HSM slot

6.3 Adding certificate to GnuPG

1. Create a master key based on the existing key.

```
# gpg --expert --full-generate-key
```

2. Select option **(14) Existing key from card**. This will list the serial number of the HSM slot and existing keys which has a corresponding certificate.

```
root@debian:~# gpg --expert --full-generate-key
gpg (GnuPG) 2.5.18; Copyright (C) 2025 g10 Code GmbH
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

Please select what kind of key you want:
   (1) RSA and RSA
   (2) DSA and Elgamal
   (3) DSA (sign only)
   (4) RSA (sign only)
   (7) DSA (set your own capabilities)
   (8) RSA (set your own capabilities)
   (9) ECC (sign and encrypt) *default*
  (10) ECC (sign only)
  (11) ECC (set your own capabilities)
  (13) Existing key
  (14) Existing key from card
  (16) ECC and Kyber
Your selection? 14
Serial number of the card: D2760001240111503131FE0FE7981111
Available keys:
   (1) 9A19491C66F9D4D0AF6800B0FC74C0ACC6A2187C Utimaco\x20IS\x20GmbH/CryptoServer/SI003001_0000//45 rsa2048
Your selection?
```

Figure 52 : GPG command to select existing key from HSM

3. Enter the number for the keys you want to use.

```
Serial number of the card: D2760001240111503131FE0FE7981111
Available keys:
   (1) 9A19491C66F9D4D0AF6800B0FC74C0ACC6A2187C Utimaco\x20IS\x20GmbH/CryptoServer/SI003001_0000//45 rsa2048
Your selection? 1

Possible actions for this RSA key:
Current allowed actions:

   (Q) Finished

Your selection?
```

Figure 53 : List existing keys on HSM through GPG & selecting key number

4. Enter “Q” then provide **key expiry**, **real name**, and **email address**. Provide slot PIN when prompted.

```
Please specify how long the key should be valid.
         0 = key does not expire
      <n>  = key expires in n days
      <n>w = key expires in n weeks
      <n>m = key expires in n months
      <n>y = key expires in n years
Key is valid for? (0) 0
Key does not expire at all
Is this correct? (y/N) y

GnuPG needs to construct a user ID to identify your key.

Real name: test@utimaco.com
Email address: test@utimaco.com
Comment:
You selected this USER-ID:
    "test@utimaco.com <test@utimaco.com>"

Change (N)ame, (C)omment, (E)mail or (O)kay/(Q)uit? O
gpg: revocation certificate stored as '/root/.gnupg/openpgp-revocs.d/FCA29EF046566DE94D1140A580C3935DD6B81017.rev'
public and secret key created and signed.

pub   rsa2048 2026-04-13 [SCEAR]
      FCA29EF046566DE94D1140A580C3935DD6B81017
uid                      test@utimaco.com <test@utimaco.com>

root@debian:~#
```

Figure 54 : Finishing GPG Key Generate Command

5. List the keys.

```
# gpg --list-keys
```

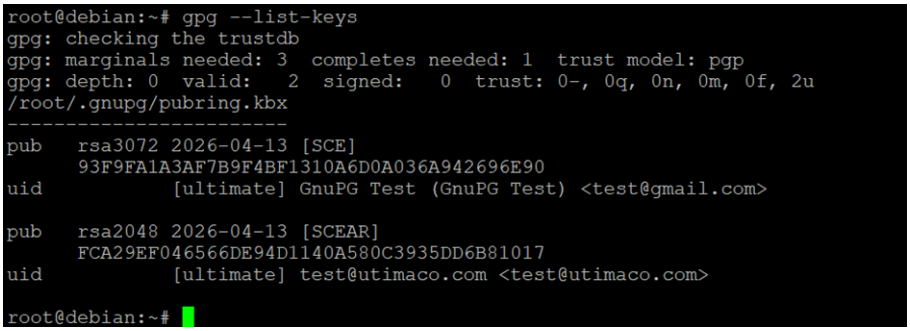

Figure 55 : GPG list keys

6.4 Signing, Encryption, Decryption and Verification with GnuPG

1. Create a sample message file.

```
# echo "Welcome to Utimaco" > message.txt
```

2. Sign the file using the key name.

```
# gpg --output message.txt.signed --sign --default-key test@utimaco.com
message.txt
```

Provide slot PIN when prompted. This will generate a signed file `message.txt.signed`.

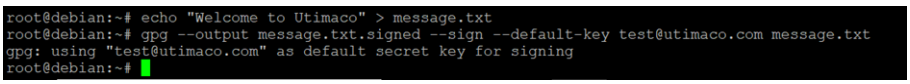

Figure 56 : Signing the file

3. Verify the file `message.txt.signed`.

```
# gpg --verify message.txt.signed
```

```
root@debian:~# gpg --verify message.txt.signed
gpg: Signature made Mon 13 Apr 2026 04:05:20 AM PDT
gpg:                using RSA key FCA29EF046566DE94D1140A580C3935DD6B81017
gpg:                issuer "test@utimaco.com"
gpg: Good signature from "test@utimaco.com <test@utimaco.com>" [ultimate]
root@debian:~#
```

Figure 57 : Verifying the signed file

4. Encrypt the file. Provide recipient user ID when prompted.

```
# gpg --output message.txt.enc --encrypt  message.txt
```

```
root@debian:~# gpg --output message.txt.enc --encrypt message.txt
You did not specify a user ID. (you may use "-r")

Current recipients:

Enter the user ID.  End with an empty line: test@utimaco.com

Current recipients:
rsa2048/80C3935DD6B81017 2026-04-13 "test@utimaco.com <test@utimaco.com>"

Enter the user ID.  End with an empty line:
```

Figure 58 : Encrypting the file

This will generate an encrypted file `message.txt.enc`.

5. Decrypt the encrypted file.

```
# gpg --output message.txt.dec --decrypt  message.txt.enc
```

Provide slot PIN when prompted. This will generate a decrypted file `message.txt.dec`.

```
root@debian:~# gpg --output message.txt.dec --decrypt message.txt.enc
gpg: encrypted with rsa2048 key, ID 80C3935DD6B81017, created 2026-04-13
      "test@utimaco.com <test@utimaco.com>"
root@debian:~#
```

Figure 59 : Decrypting the file

6. Verify the content of the original file.

```
# cat message.txt.dec
```

Figure 60 : Original content of the file

6.5 RPM Signing and Verification with GnuPG

6.5.1 RPM Signing

1. Create RPM build directories.

```
mkdir -p ~/rpmbuild/{BUILD,RPMS,SOURCES,SPECS,SRPMS}
```

2. Create a test source file.

```
echo "RPM signing test file" > ~/rpmbuild/SOURCES/hello.txt
```

3. Create a minimal SPEC file.

```
cat <<EOF > ~/rpmbuild/SPECS/hello.spec
Name:           hello
Version:        1.0
Release:        1
Summary:        Test RPM for signing

License:        GPL
BuildArch:      noarch

%description
Test RPM created to validate GPG/HSM-based RPM signing.

%install
mkdir -p %{buildroot}/usr/share/hello
cp %{_sourcedir}/hello.txt %{buildroot}/usr/share/hello/

%files
/usr/share/hello/hello.txt
EOF
```

4. Create a file `/root/.rpmmacros` in the user's home directory and add the following content in it.

```
%_signature gpg
%_gpg_path /root/.gnupg
%_gpg_name test@utimaco.com
%_gpg /usr/local/bin/gpg
%_gpg_sign_cmd %{_gpg} gpg --force-v3-sigs --batch --verbose --no-armor -no-
secmem-warning -u "%{_gpg_name}" -sbo %{_signature_filename} --digestalgo
filename}
```

Figure 61 : Content of rpmmacros file

Here:

- `/root/.gnupg` is the base directory for gnupg.
- test@utimaco.com is the key name.
- `/usr/local/bin/gpg` is the path for gpg.
- `%{_gpg} gpg --force-v3-sigs --batch --verbose --no-armor --no-secmem-warning -u "%{_gpg_name}" -sbo %{_signature_filename} --digest-algo filename}` is the gpg command that will be used for signing rpm.

5. Sign the file using the command below.

```
# rpm --addsign <rpm_file>
```

Provide the slot PIN when prompted.

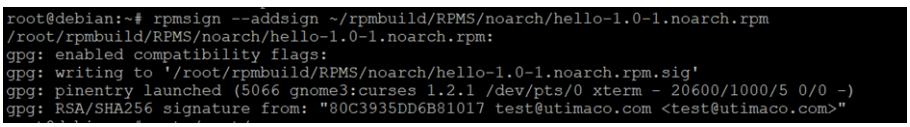

Figure 62 : RPM signing

6. If you want to sign it again, run the below command. Provide the slot PIN when prompted.

```
# rpm --resign <rpm_file>
```

6.5.2 Signed RPM Verification

1. Export the public key to a file.

```
# gpg --export --armor test@utimaco.com > gpgpub.key
```

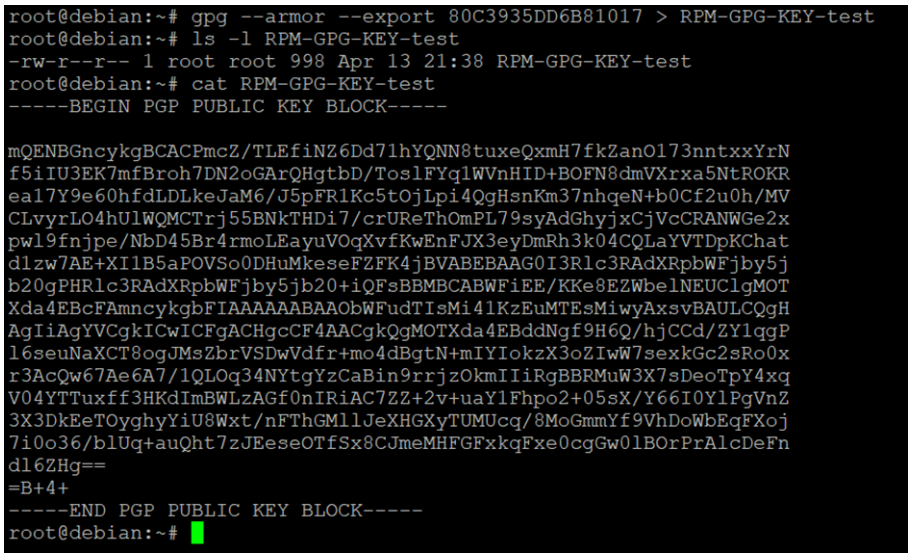

Figure 63 : Exporting the public key to a file

2. Import the public key.

```
# rpm --import RPM-GPG-KEY-test
```

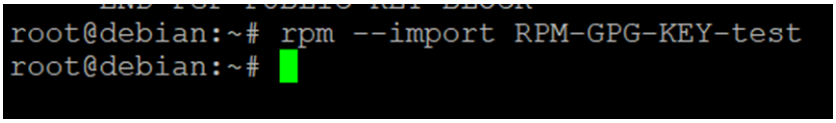

Figure 64 : Importing the public key to rpm db

3. Verify the signature of the signed rpm.

```
# rpm --checksig <signed_rpm_file>
```

```
[root@rk ~]# rpm --checksig shim-15-8.el7.src.rpm
shim-15-8.el7.src.rpm: digests signatures OK
[root@rk ~]#
```

Figure 65 : Verifying the signed rpm file

4. Verify the signing information.

```
# rpm -qpi ~/rpmbuild/RPMS/noarch/hello-1.0-1.noarch.rpm
```

The signature field contains the signing information.

```
root@debian:~# rpm -qpi ~/rpmbuild/RPMS/noarch/hello-1.0-1.noarch.rpm
Name        : hello
Version     : 1.0
Release     : 1
Architecture: noarch
Install Date: (not installed)
Group       : Unspecified
Size        : 22
License     : GPL
Signature   : RSA/SHA256, Mon 13 Apr 2026 09:05:51 PM PDT, Key ID 80c3935dd6b81017
Source RPM  : hello-1.0-1.src.rpm
Build Date  : Mon 13 Apr 2026 08:54:12 PM PDT
Build Host  : debian.debian
Summary     : Test RPM for signing
Description :
Test RPM created to validate GPG/HSM-based RPM signing.
root@debian:~#
```

Figure 66 : Verifying the signed rpm file signature information

This completes the Integration of GnuPG with u.trust GP HSM.

7 Troubleshooting

Error	Diagnosis
LoginUser= failed: 05.12.2021 23:45:45 src/p11adm_R2.c[429] p11_login: C_Login [type=1] returned Error 0x00000102 (CKR_USER_PIN_NOT_INITIALIZED)	PKCS#11 slot is not initialized.
The CryptoServer PKCS#11 Library R3 is not initialized. Error CKR_CRYPTOKI_NOT_INITIALIZED occurred.	PKCS#11 slot is not initialized.

Table 6: List of errors and their diagnoses

8 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 u.trust GP HSM product documentation is also available at the Utimaco IS GmbH website: https://utimaco.com/.

9 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.

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.

10 Appendices

10.1 References

Reference	Title/Company	Document No.
[CSADMIN]	CryptoServer – csadm Manual/Utimaco IS GmbH	2009-0003
[CSTrSh]	CryptoServer Troubleshooting/Utimaco IS GmbH	M011-0008-en
[CSADMIN2]	CryptoServer_csadm_Manual_Systemadministrators.pdf	2009-0003
[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: References

10.2 Command Summary

Commands Used	Purpose
`mkdir -p /etc/utimaco/bin`	Create a directory for Utimaco binaries
`mkdir /etc/utimaco/lib`	Create a directory to store PKCS#11 library
`cp ~/path_to_application_folder/lib/ libcs_pkcs11_R3.so /etc/utimaco/lib`	Copy Utimaco PKCS#11 library to system path

Commands Used	Purpose
`cd ~/path_to_application_folder`	Navigate to application directory
`cp csadm p11tool2 /etc/utimaco/bin`	Copy PKCS#11 admin and utility tools
`chmod +x /etc/utimaco/bin/csadm /etc/utimaco/bin/p11tool2`	Make tools executable
`mkdir /etc/utimaco/PKCS11`	Create a directory for PKCS#11 configuration
`cd <install directory>/Software/Linux/x86-64/Crypto_APIs/PKCS11_R3/sample # cp cs_pkcs11_R3.cfg /etc/utimaco/PKCS11 # cd /etc/utimaco/PKCS11`	Copy the PKCS#11 configuration file
`./p11tool2 slot=<slot_no> Label=<token_label> Login=ADMIN,ADMIN.key InitToken=<SO_PIN>`	Initialize the HSM token with Security Officer (SO) PIN
`./p11tool2 slot=0 LoginSO=<SO_PIN> InitPin=<CryptoUser_PIN>`	Set the Crypto User PIN
`wget https://www.gnupg.org/ftp/gcrypt/libgpg-error/libgpg-error-1.59.tar.bz2`	Download `libgpg-error` source package
`tar -xjf libgpg-error-1.59.tar.bz2`	Extract the downloaded package

Commands Used	Purpose
`export PATH=/usr/local/bin:$PATH`	Update the execution path
`export LD_LIBRARY_PATH=/usr/local/lib`	Set the library path for runtime linking
`gpg --version`	Verify GnuPG installation
`sudo apt update`	Refresh the package index
`sudo apt install -y libssl-dev`	Install OpenSSL development libraries
`pkg-config --modversion libcrypto`	Check OpenSSL (libcrypto) version
`sudo apt install -y libpkcs11-helper1-dev`	Install the PKCS#11 helper library
`dpkg -L libpkcs11-helper1-dev | grep pkgconfig`	grep pkgconfig`
`export PKG_CONFIG_PATH=/usr/lib/x86_64-linux-gnu/pkgconfig`	Set pkg-config search path
`pkg-config --modversion libpkcs11-helper-1`	Confirm PKCS#11 helper installation
`gpg --list-keys`	List available GPG keys

Commands Used	Purpose
`cp /usr/local/src/gnupg-pkcs11-scd-0.11.0/gnupg-pkcs11-scd/gnupg-pkcs11-scd.conf.example /root/.gnupg/gnupg-pkcs11-scd.conf`	Copy PKCS#11 smart card daemon config
`./p11tool2 Slot=0 LoginUser=Gnupg1234 PubKeyAttr=CKA_LABEL="GPGPublicKey",CKA_MODULUS_BITS=2048,CKA_ID=0x45 PrvKeyAttr=CKA_LABEL="GPGPrivateKey",CKA_EXTRACTABLE=CK_TRUE,CKA_ID=0x45 GenerateKeyPair=RSA`	Generate an RSA key pair inside HSM
`./p11tool2 Slot=0 LoginUser=Gnupg1234 ListObjects`	List objects stored in the HSM
`apt install -y libengine-pkcs11-openssl`	Install the OpenSSL PKCS#11 engine
`echo "Welcome to Utimaco" > message.txt`	Create a test message file
`gpg --output message.txt.signed --sign --default-key test@utimaco.com message.txt`	Digitally sign a file using GPG
`gpg --verify message.txt.signed`	Verify the digital signature
`gpg --output message.txt.enc --encrypt message.txt`	Encrypt a file using GPG
`gpg --output message.txt.dec --decrypt message.txt.enc`	Decrypt an encrypted file

Commands Used	Purpose
`cat message.txt.dec`	View decrypted content
`gpg --export --armor test@utimaco.com > gpgpub.key`	Export a public key
`rpm --import RPM-GPG-KEY-test`	Import an RPM GPG key
`rpm -qpi ~/rpmbuild/RPMS/noarch/ hello-1.0-1.noarch.rpm`	Query RPM package information
`openssl engine dynamic \` `-pre SO_PATH:/opt/openssl-1.1.1/lib/engines-1.1/ pkcs11.so \` `-pre ID:pkcs11 \` `-pre LIST_ADD:1 \` `-pre LOAD \` `-pre MODULE_PATH:/etc/utimaco/lib/libcs_pkcs11_R3.so`	Load PKCS#11 engine dynamically in OpenSSL
`openssl req -new -x509` `-engine pkcs11 \` `-keyform engine \` `-key "pkcs11:id=%45;type=private" \` `-sha256 \` `-days 365 \` `-out test.pem \` `-subj "/CN=test.utimaco.com/O=Integration"`	Generate a CSR/ certificate using a PKCS#11 key

Commands Used	Purpose

`openssl x509 \`
`-in test.pem \`
`-out test.der \`
`-outform DER`
	Convert the certificate to DER format

`pkcs11-tool \`
`--module /etc/utimaco/lib/libcs_pkcs11_R3.so \`
`-l \`
`--slot 0 \`
`--write-object test.der \`
`--type cert \`
`--id 45 \`
`--label "GPGCertificate"`
	Upload the certificate to HSM

`gpg --expert --full-generate-key`	Generate a GPG key (advanced options)

Table 8: List of commands used