Ethereum Blockchain: Integration Guide

THALES LUNA HSM AND DPOD LUNA CLOUD HSM

Document Information

Document Part Number	007-000189-001
Revision	B
Release Date	7 December 2022

Trademarks, Copyrights, and Third-Party Software

CONTENTS

Overview ... 4
Understanding Ethereum Blockchain Network ... 4
Certified Platforms ... 5
Prerequisites ... 5
Configure Luna HSM ... 5
Configure Luna Cloud HSM Service ... 6
Set up Go-Ethereum ... 7
Configuring Luna HSM to use with Ethereum Blockchain ... 7
Configure Luna Network HSM for Ethereum Blockchain ... 7
Configure Luna Cloud HSM for Ethereum Blockchain ... 10
Integrating Ethereum Blockchain with Luna HSM ... 11
Integrate Ethereum with Luna HSM using BIP32 keys ... 12
Integrate Ethereum with Luna HSM using ECDSA keys ... 17
Contacting Customer Support ... 25
Customer Support Portal ... 25
Telephone Support ... 25

Overview

This guide demonstrates using a Luna HSM or Luna Cloud HSM in Ethereum Blockchain to protect the Ethereum account private keys.

In Ethereum, an account is the public/private key pair file that serves as the identity on the Blockchain. The HSM wallet allows you to generate and secure ECDSA/BIP32 key pairs on Luna HSM, following which you can use Luna HSM to sign transactions. The HSM wallet uses a PKCS#11 API to communicate with Luna HSM. Each PKCS#11 partition corresponds to a different HSM wallet.

The benefits of using Luna HSMs to generate the private keys for Ethereum Blockchain Accounts include:

> Secure generation, storage and protection of the encryption keys on FIPS 140-2 level 3 validated hardware

> Full life cycle management of the keys

> Access to secure audit trail

NOTE: Access to secure audit trail is available only for Luna Network HSM.

> Using cloud services with confidence.

> Significant performance improvements by off-loading cryptographic operations from application servers.

Understanding Ethereum Blockchain Network

Ethereum is an open-source, public, blockchain-based distributed computing platform and operating system featuring smart contract (scripting) functionality. Ether is the fundamental cryptocurrency for operation of Ethereum, which thereby provides a public distributed ledger for transactions. It is used to pay for gas, a unit of computation used in transactions and other state transitions. Ether can be transferred between accounts and used to compensate participant mining nodes for computations performed.

Go Ethereum is the official golang implementation of Ethereum protocol. Geth is the main Ethereum CLI client. It is the entry point into the Ethereum network (main, test or private net), capable of running as a full node (default), archive node (retaining all historical state), or a light node (retrieving data live). It can be used by other processes as a gateway into the Ethereum network via JSON RPC endpoints exposed on top of HTTP, Web Socket and/or IPC transports.

The proof of work (PoW) algorithm used in Ethereum is Ethash (a modified version of the Dagger-Hashimoto algorithm). Ethash PoW is memory hard, making it ASIC resistant. Memory hardness is achieved with a proof of work algorithm that requires choosing subsets of a fixed resource dependent on the nonce and block header. This resource (a few gigabyte size data) is called a DAG.

Consult Go-Ethereum documentation or more information on its implementation and working.

Certified Platforms

This integration is certified on the following platforms:

- Certified platforms on Luna HSM
- Certified platforms on Luna Cloud HSM

Certified platforms on Luna HSM

HSM Type	Platforms Tested
Luna HSM	RHEL Ubuntu

Luna HSM: Luna HSM appliances are purposefully designed to provide a balance of security, high performance, and usability that makes them an ideal choice for enterprise, financial, and government organizations. Luna HSMs physically and logically secure cryptographic keys and accelerate cryptographic processing. The Luna HSM on premise offerings include the Luna Network HSM, Luna PCIe HSM, and Luna USB HSMs. Luna HSMs are also available for access as an offering from cloud service providers such as IBM Cloud HSM and AWS CloudHSM Classic.

Certified platforms on Luna Cloud HSM

HSM Type	Platforms Tested
Luna Cloud HSM	RHEL Ubuntu

Luna Cloud HSM: Luna Cloud HSM services provide on-demand, cloud-based storage, management, and generation of cryptographic keys through a simple graphical user interface. With Luna Cloud HSM, security is simple, cost effective, and easy to manage because there is no hardware to buy, deploy, and maintain. As an Application Owner, you click and deploy services, generate usage reports, and maintain only those services that you need.

Prerequisites

Before you proceed with the any of the integrations described in this document, complete the following tasks:

Configure Luna HSM

Configure Luna Cloud HSM Service

Set up Go-Ethereum

Configure Luna HSM

To configure Luna HSM with Go-Ethereum:

1. Verify that the HSM is set up, initialized, provisioned, and ready for deployment.
2. Create a partition on the HSM that will be later used as HSM wallet.
3. If using a Luna Network HSM, register a client for the system and assign the client to the partition to create an NTLS connection. Initialize the Crypto Officer and Crypto User roles for the registered partition.

> **NOTE:** Refer to Configuring Luna HSM for Ethereum Blockchain for detailed steps on creating NTLS connection, initializing the partitions, and assigning various user roles.

Set up Luna HSM High-Availability Group

Refer to the Luna HSM documentation for HA steps and details regarding configuring and setting up two or more HSM boxes on host systems. You must enable the HAOnly setting in HA for failover to work so that if the primary goes down due to any reason, all calls get automatically routed to the secondary until the primary recovers and starts up.

Set up Luna HSM in FIPS Mode

> **NOTE:** This setting is not required for Luna HSM Universal Client. This setting is applicable only for Luna HSM Client 7.x.

Under FIPS 186-3/4, the RSA methods permitted for generating keys are 186-3 with primes and 186-3 with aux primes. This means that RSA PKCS and X9.31 key generation is no longer approved for operation in a FIPS-compliant HSM. If you are using Luna HSM in FIPS mode, you have to make the following change in the configuration file:

```
Misc = {
RSAKeyGenMechRemap = 1;
}
```

The above setting redirects the older calling mechanism to a new approved mechanism when Luna HSM is in FIPS mode.

Configure Luna Cloud HSM Service

If you are using Luna Cloud HSM, you have the following configuration options:

- Standalone Cloud HSM service using minimum client package
- Standalone Cloud HSM service using full Luna client package
- Luna HSM and Luna Cloud HSM service in hybrid mode

> **NOTE:** Refer to Configuring Luna Cloud HSM for Ethereum Blockchain for detailed steps on provisioning Luna Cloud HSM service, initializing the partition, and assigning various user role.
>
> **NOTE:** Follow the Luna Cloud HSM documentation for detailed steps for creating service, client, and initializing various user roles.
>
> **NOTE:** Luna Client v10.x or higher is required for configuring Luna HSM device and Luna Cloud HSM service in hybrid mode.

Luna Cloud HSM Service in FIPS mode

Luna Cloud HSM service operates in both FIPS and non-FIPS mode. If your organization requires non-FIPS algorithms for your operations, ensure you enable the **Allow non-FIPS approved algorithms** check box when configuring your Cloud HSM service. The FIPS mode is enabled by default. Refer to the Mechanism List in the SDK Reference Guide for more information about available FIPS and non-FIPS algorithms.

Set up Go-Ethereum

Complete the following processes to install and configure Go-Ethereum on a Linux Operating System.

Install prerequisite libraries

Install the following components on the Linux operating system:

```
# sudo yum install git python3-pip libtool-ltdl-devel          (RHEL)
# sudo apt-get install git alien python3-pip libltdl-dev       (Ubuntu)
```

Install and setting up Golang

Ethereum Blockchain uses the Go programming language for many of its components. Install the golang using the steps provided by Golang official site.

Installation steps: https://golang.org/doc/install

Download binaries: https://golang.org/dl/

Install and set up Docker and Docker-compose

Docker and Docker-compose need to be installed on the platform on which you are operating.

Follow the instructions in the following URL to install the docker:

Installation steps: https://docs.docker.com/engine/install/

Install the docker-compose:

```
# sudo pip3 install docker-compose
```

> **NOTE:** It is always recommended to use the latest version available.

Install and configure GO-Ethereum

Installation and configuration of Go-Ethereum for Luna HSM is provided in the respective section of this integration guide.

Configuring Luna HSM to use with Ethereum Blockchain

Luna HSM provides strong protection of secure assets, including keys, and should be considered a best practice when building systems based on Ethereum Blockchain. This section provides instructions for configuring Luna HSM for Go-Ethereum. Follow the instructions provided below as per the HSM you are using:

> Configure Luna Network HSM for Ethereum Blockchain

> Configure Luna Cloud HSM for Ethereum Blockchain

Configure Luna Network HSM for Ethereum Blockchain

To configure Luna HSM ensure the following:

1. Ensure that the HSM is setup, initialized, provisioned, and ready for deployment.
2. Create two HSM partitions, for example TPA01 and TPA02, which will be used by geth1 and geth2 node of Ethereum Blockchain.

> **NOTE:** You can use same HSM or different HSM for both partitions. Partition name can be any user defined name.

3. Create the following directories on the client machine:

```
# mkdir -p /usr/local/lunaclient
# mkdir -p /etc/geth/lunaconf1
# mkdir -p /etc/geth/lunaconf2
```

4. Extract the Luna Client minimal package installer in folder /usr/local/lunaclient.

```
# tar -C /usr/local/lunaclient --strip 1 -xvf LunaClient-Minimal-<release_version>.x86_64.tar
```

5. Set the ChrystokiConfigurationPath variable to point to the directory /etc/geth/lunaconf1.

```
# export ChrystokiConfigurationPath=/etc/geth/lunaconf1
```

6. Copy the Chrystoki-template.conf to the /etc/geth/lunaconf1 directory.

```
# cp /usr/local/lunaclient/Chrystoki-template.conf /etc/geth/lunaconf1/Chrystoki.conf
```

7. Modify the Chrystoki2 section of /etc/geth/lunaconf1/Chrystoki.conf to:

```
Chrystoki2 = {
    LibUNIX = /usr/local/lunaclient/libs/64/libCryptoki2.so;
    LibUNIX64 = /usr/local/lunaclient/libs/64/libCryptoki2_64.so;
}
```

8. Modify the LunaSA Client section of /etc/geth/lunaconf1/Chrystoki.conf and update the following:

> **NOTE:** Do not update/delete any other flags of this section.

```
LunaSA Client = {
   SSLConfigFile = /usr/local/lunaclient/openssl.cnf;
   ClientPrivKeyFile = /etc/geth/lunaconf1/;
   ClientCertFile = /etc/geth/lunaconf1/;
   ServerCAFile = /etc/geth/lunaconf1/CAFile.pem;
}
```

9. Modify the Secure Trusted Channel section of /etc/geth/lunaconf1/Chrystoki.conf file as follows:

> **NOTE:** Secure Trusted Channel section of /etc/geth/lunaconf1/Chrystoki.conf can be completely removed if STC is not enabled.

```
Secure Trusted Channel = {
   ClientTokenLib = /usr/local/lunaclient/libs/64/libSoftToken.so;
   SoftTokenDir = /usr/local/lunaclient/stc/token;
   ClientIdentitiesDir = /usr/local/lunaclient/client_identities;
   PartitionIdentitiesDir = /usr/local/lunaclient/partition_identities;
}
```

10. Copy the server certificate from Luna HSM to current directory.

```
# scp admin@<HSM-IP>:server.pem .
```

11. Add the Luna HSM server certificate on client.

```
# /usr/local/lunaclient/bin/64/vtl addServer -n <HSM-IP> -c server.pem
```

12. Create the client certificate.

```
# /usr/local/lunaclient/bin/64/vtl createCert -n geth1
```

13. Transfer the client certificate to the Luna HSM.

```
# scp /etc/geth/lunaconf1/geth1.pem admin@<HSM-IP>:
```

14. Log in to Luna HSM and register the client and assign TPA01 partition to the client.
15. Use the lunacm utility from the client machine to initialize the partition and Crypto Officer role.

```
[root@lkb-ms lunaclient]# /usr/local/lunaclient/bin/64/lunacm
lunacm (64-bit) v10.4.0-417. Copyright (c) 2021 SafeNet. All rights reserved.


        Available HSMs:

        Slot Id ->              0
        Label ->                TPA01
        Serial Number ->        1312109862201
        Model ->                LunaSA 7.7.1
        Firmware Version ->     7.7.1
        Bootloader Version ->   1.1.2
        Configuration ->        Luna User Partition With SO (PW) Key Export With Cloning Mode
        Slot Description ->     Net Token Slot
        FM HW Status ->         Non-FM


        Current Slot Id: 0
```

16. For PED-authenticated HSM, enable partition policies 22 and 23 to allow activation and auto-activation.

> **NOTE:** Refer to Luna HSM documentation for detailed steps on creating NTLS connection, initializing the partitions, and assigning various user roles.

17. Repeat steps 5 to 16 for HSM/partition2 replacing lunaconf1, geth1, and TPA01 with lunaconf2, geth2, and TPA02, respectively, at each step.

> **NOTE:** Replace TPA01 and TPA02 with the actual partition label.

Configure Luna Cloud HSM for Ethereum Blockchain

To configure Luna Cloud HSM:

1. Create two Luna Cloud HSM Services in Data Protection on Demand platform.

> **NOTE:** Consult Data Protection on Demand Application Owner Quick Start Guide for more information about provisioning the HSM on Demand service and initializing the service client.

2. For each service, create a client and download client zip files to the host system.
3. Make the following two directories on client machine:

```
# mkdir -p /etc/geth/lunaconf1
# mkdir -p /etc/geth/lunaconf2
```

4. Unzip the downloaded client zip files in to directories lunaconf1 and lunaconf2, respectively.

```
# unzip /home/setup-<client1>.zip -d /etc/geth/lunaconf1
# unzip /home/setup-<client2>.zip -d /etc/geth/lunaconf2
```

5. Go to the lunaconf1 and lunaconf2 directories and extract cvclient-min.tar file in both the directories.

```
# cd /etc/geth/lunaconf1
# tar -xvf cvclient-min.tar
# cd /etc/geth/lunaconf2
# tar -xvf cvclient-min.tar
```

> **Note:** Follow the instructions provided in the DPOD Application Owner Quick Start Guide and initialize both the partitions, Security Officer and Crypto Officer.

6. Verify that each partition is visible from the client via the lunacm utility after initializing the partition and required roles.

```
[root@lkb-ms lunaconf2]# cd /etc/geth/lunaconf1
[root@lkb-ms lunaconf1]# export ChrystokiConfigurationPath=/etc/geth/lunaconf1
[root@lkb-ms lunaconf1]# ./bin/64/lunacm
lunacm (64-bit) v10.5.0-470. Copyright (c) 2022 SafeNet. All rights reserved.


        Available HSMs:

        Slot Id ->              3
        Label ->                TPA01
        Serial Number ->        1393032470862
        Model ->                Cryptovisor7
        Firmware Version ->     7.3.0
        CV Firmware Version ->  2.0.1
        Plugin Version ->       Cloud 2.2.0-740
        Configuration ->        Luna User Partition With SO (PW) Signing With Cloning Mode
        Slot Description ->     Net Token Slot
        FM HW Status ->         FM Not Supported


        Current Slot Id: 3


lunacm:>exit
[root@lkb-ms lunaconf1]# cd /etc/geth/lunaconf2
[root@lkb-ms lunaconf2]# export ChrystokiConfigurationPath=/etc/geth/lunaconf2
[root@lkb-ms lunaconf2]# ./bin/64/lunacm
lunacm (64-bit) v10.5.0-470. Copyright (c) 2022 SafeNet. All rights reserved.


        Available HSMs:

        Slot Id ->              3
        Label ->                TPA02
        Serial Number ->        1393032470863
        Model ->                Cryptovisor7
        Firmware Version ->     7.3.0
        CV Firmware Version ->  2.0.1
        Plugin Version ->       Cloud 2.2.0-740
        Configuration ->        Luna User Partition With SO (PW) Signing With Cloning Mode
        Slot Description ->     Net Token Slot
        FM HW Status ->         FM Not Supported


        Current Slot Id: 3


lunacm:>
```

Integrating Ethereum Blockchain with Luna HSM

When Go-Ethereum is integrated with Luna HSM, it generates either BIP32 or ECDSA keys for signing. Luna HSM protects the account signing key and only an authenticated user account can access the signing keys to perform signed transaction in Blockchain network.

This section describes how to use Luna HSM to generate account signing keys and sign transactions using Luna HSM generated keys.

> Integrate Ethereum with Luna HSM using BIP32 keys

> Integrate Ethereum with Luna HSM using ECDSA keys

> **Note:** Luna HSM and Luna Cloud HSM supports **BIP32** mechanism in **Non-FIPS** mode only. If your HSM is in FIPS mode you can only able to use ECDSA keys.

Integrate Ethereum with Luna HSM using BIP32 keys

Luna HSM or Luna Cloud HSM must be in Non-FIPS mode to use BIP32 mechanism as this mechanism is not supported in FIPS mode.

To integrate Ethereum with Luna HSM using BIP32 keys:

1. Ensure that the go executable is in the PATH.

   `# export PATH=/usr/local/go/bin:$PATH`

2. Clone the Go-Ethereum git repository on the client.

   `# git clone https://github.com/ThalesGroup/go-ethereum`

3. Change directory to the cloned repo.

   `# cd go-ethereum`

4. Build the Geth binary.

   `# make geth`

5. Build a docker image for a Geth node.

   `# cd example`

   `# make build`

6. Ensure that Docker images are built successfully.

   `# docker images`

```
[root@lkb-ms example]# docker images
REPOSITORY    TAG       IMAGE ID       CREATED          SIZE
geth          latest    aa20b3b73113   7 seconds ago    330MB
ubuntu        xenial    b6f507652425   14 months ago    135MB
[root@lkb-ms example]#
```

7. Create a Geth config file in go-ethereum/example directory.

   `#  ../build/bin/geth --networkid 8000 dumpconfig > config.toml`

```
[root@lkb-ms example]# ../build/bin/geth --networkid 8000 dumpconfig > config.toml
INFO [11-16|17:08:30.545] Maximum peer count                       ETH=50 LES=0 total=50
INFO [11-16|17:08:30.546] Smartcard socket not found, disabling    err="stat /run/pcscd/pcscd.comm: no such file or directory"
INFO [11-16|17:08:30.553] Set global gas cap                       cap=50,000,000
[root@lkb-ms example]#
```

   For Luna Cloud HSM:

   Create two geth config files in `go-ethereum/example` directory as we have two different service client.

   `#  ../build/bin/geth --networkid 8000 dumpconfig > config1.toml`

   `#  ../build/bin/geth --networkid 8000 dumpconfig > config2.toml`

8. Add the following flags to the [Node] section in config.toml:

   `NoPKCS11BIP32 = false`

   `PKCS11Lib = "/usr/local/lunaclient/libs/64/libCryptoki2.so"`

```
[Node]
NoPKCS11BIP32 = false
PKCS11Lib = "/usr/local/lunaclient/libs/64/libCryptoki2.so"
DataDir = "/root/.ethereum"
IPCPath = "geth.ipc"
```

   <u>For Luna Cloud HSM:</u>

   In config1.toml:

   `NoPKCS11BIP32 = false`

   `PKCS11Lib = "/etc/geth/lunaconf1/libs/64/libCryptoki2.so"`

   In config2.toml:

   `NoPKCS11BIP32 = false`

   `PKCS11Lib = "/etc/geth/lunaconf2/libs/64/libCryptoki2.so"`

9. Create two ethash directories for the DAG in go-ethereum/example directory.

   `# mkdir ethash1`

   `# mkdir ethash2`

10. Initialize the data directories:

   `# make clean`

   `# make init`

11. If using Luna Cloud HSM, update the docker-compose.yaml under go-ethereum/example directory. Make the following changes:

   > **Note:** If using Luna HSM, Skip this step as no update is required for Luna HSM. These modification in docker-compose.yaml is required only for Luna Cloud HSM.

   a. Remove the following line from [volumes] section of geth1.

      `- /usr/local/lunaclient:/usr/local/lunaclient`

   b. Make the following change in the volume section of geth1:

      Update:

      `./config.toml:/etc/geth/config.toml`

      To

      `./config1.toml:/etc/geth/config.toml`

   c. Remove the following line from [volumes] section of geth2.

      `- /usr/local/lunaclient:/usr/local/lunaclient`

d. Make the following change in the volume section of geth2:

Update:

```
./config.toml:/etc/geth/config.toml
```

To

```
./config2.toml:/etc/geth/config.toml
```

After making these changes, ensure that docker-compose.yaml looks like the following:

```
  geth1:
    container_name: geth1
    image: geth
    working_dir: "/"
    environment:
      - ChrystokiConfigurationPath=/etc/geth/lunaconf1
    volumes:
      - /etc/geth/lunaconf1:/etc/geth/lunaconf1
      - ./config1.toml:/etc/geth/config.toml
      - ./data1:/etc/geth/data
      - ./ethash1:/etc/geth/ethash
    ports:
      - "30303"
    networks:
      vpcbr:
        ipv4_address: 10.8.0.3



  geth2:
    container_name: geth2
    image: geth
    working_dir: "/"
    environment:
      - ChrystokiConfigurationPath=/etc/geth/lunaconf2
    volumes:
      - /etc/geth/lunaconf2:/etc/geth/lunaconf2
      - ./config2.toml:/etc/geth/config.toml
      - ./data2:/etc/geth/data
      - ./ethash2:/etc/geth/ethash
    ports:
      - "30303"
    networks:
      vpcbr:
        ipv4_address: 10.8.0.4
```

12. Create two docker containers (geth1 and geth2) running Geth image.

```
# docker-compose up
```

13. Open a new terminal and attach to node 1.

```
# ./console.sh 1
```

14. Initially the wallet is closed. You can check the closed status by running following command in console 1.

```
# personal.listWallets
```

```
> personal.listWallets
[{
    status: "Closed",
    url: "hsm://TPA01"
}]
```

15. Open the wallet in console 1. Provide the partition password when prompted.

```
# personal.openWallet("hsm://<partition1>")
```

```
> personal.openWallet("hsm://TPA01")
Please enter the partition password:
null
>
```

When you open the wallet for the first time, it will generate master seed and derive a master key pair. A child key pair will be derived for path “m/44'/60'/0'/0/0” by the Self-Derive background task. Self-derivation will run in the background automatically deriving new keys starting at path "m/44'/60'/0'/0/0". To see account you can run the `personal.listWallets` command.

```
> personal.listWallets
[{
    accounts: [{
        address: "0x495f6e0fce2e950a4f29d6a006cf925e0b95dd52",
        url: "hsm://TPA01/m/44'/60'/0'/0/0"
    }],
    status: "Open",
    url: "hsm://TPA01"
}]
```

The background thread monitors the block chain to see whether the last key derived is part of a transaction and added to a block. If that is the case, the background thread will derive the next account. For example, "m/44'/60'/0'/1/0" when listing accounts, as illustrated in the next step.

16. Derive an account:

```
# personal.deriveAccount("hsm://<partition1>", "m/44'/60'/0'/1/0", true)
```

```
> personal.deriveAccount("hsm://TPA01", "m/44'/60'/0'/1/0", true)
{
  address: "0xa05070cc2ea2cd364c85fb85ab32e7f60dfcbb06",
  url: "hsm://TPA01/m/44'/60'/0'/1/0"
}
```

Here, **m/44'/60'/0'/1/0** refers to the path of the child key and **true** means to keep track of the account for signing later on.

> **NOTE:** The path “m/44'/60'/0'/1/0”, defines the indexes to be used when deriving child keys in a BIP32-HD Wallet. You can also choose other index values. For more information follow this link https://github.com/bitcoin/bips/blob/master/bip-0032.mediawiki

17. Set the coin base to the new derived account.

```
# miner.setEtherbase("<derive account address>")
```

Where derive account is address obtained from the previous step. For Example:

```
# miner.setEtherbase("0xa05070cc2ea2cd364c85fb85ab32e7f60dfcbb06")
```

```
> miner.setEtherbase("0xa05070cc2ea2cd364c85fb85ab32e7f60dfcbb06")
true
>
```

18. Fund the account by starting the miner:

```
# miner.start()
```

```
> miner.start()
null
```

19. Wait for "Generating DAG in progress" to get completed. The activity can be seen in the "docker-compose up" terminal. This can take some time to complete. Wait for some block to be mined and then proceed to next step.

```
geth1    | INFO [11-17|13:59:15.439] Generating DAG in progress          epoch=0 percentage=94 elapsed=3m17.913s
geth1    | INFO [11-17|13:59:17.523] Generating DAG in progress          epoch=0 percentage=95 elapsed=3m19.997s
geth1    | INFO [11-17|13:59:19.591] Generating DAG in progress          epoch=0 percentage=96 elapsed=3m22.065s
geth1    | INFO [11-17|13:59:21.745] Generating DAG in progress          epoch=0 percentage=97 elapsed=3m24.219s
geth1    | INFO [11-17|13:59:23.940] Generating DAG in progress          epoch=0 percentage=98 elapsed=3m26.414s
geth1    | INFO [11-17|13:59:26.162] Generating DAG in progress          epoch=0 percentage=99 elapsed=3m28.636s
```

```
INFO [11-17|14:13:52.033] 🔨 mined potential block          number=252 hash=5e4564..ef2307
INFO [11-17|14:13:52.034] Commit new sealing work          number=253 sealhash=b53422..b4b5c7 uncles=0 txs=0 gas=0 fees=0 elapsed="101.764µs"
INFO [11-17|14:13:52.034] Commit new sealing work          number=253 sealhash=b53422..b4b5c7 uncles=0 txs=0 gas=0 fees=0 elapsed="176.809µs"
INFO [11-17|14:13:52.294] Successfully sealed new block    number=253 sealhash=b53422..b4b5c7 hash=130ec4..027ab8 elapsed=260.582ms
INFO [11-17|14:13:52.294] 🔗 block reached canonical chain  number=246 hash=802f05..2d3f57
```

20. Check the balance for drive account address.

```
# eth.getBalance("<derive account address>")
```

For example :

```
# eth.getBalance("0xa05070cc2ea2cd364c85fb85ab32e7f60dfcbb06")
```

You should be able to see some ether in this account.

```
> eth.getBalance("0xa05070cc2ea2cd364c85fb85ab32e7f60dfcbb06")
705000000000000000000
>
```

21. Stop the miner after some blocks have been mined.

```
# miner.stop()
```

```
> miner.stop()
null
>
```

22. Derive another account:

```
# personal.deriveAccount("hsm://<partition1>", "m/44'/60'/0'/1/1", true)
```

```
> personal.deriveAccount("hsm://TPA01", "m/44'/60'/0'/1/1", true)
{
  address: "0x3f1947217ee231e02d203a81ebeee73db3248435",
  url: "hsm://TPA01/m/44'/60'/0'/1/1"
}
```

23. Send funds from the first account to the second account:

```
# eth.sendTransaction({from:sender, to:receiver, value: amount})
```

For example:

```
# eth.sendTransaction({from:"0xa05070cc2ea2cd364c85fb85ab32e7f60dfcbb06",
to:"0x3f1947217ee231e02d203a81ebeee73db3248435", value: web3.toWei(0.5,
"ether")})
```

```
> eth.sendTransaction({from:"0xa05070cc2ea2cd364c85fb85ab32e7f60dfcbb06", to:"0x3f1947217ee231e02d203a81ebeee73db3248435", value: web3.toWei(0.5, "ether")})
"0x7294ddf2b69f0cc84cae00ac4ab6f7089fa80442babf569b7f7ccc4a1c3c0314"
>
```

24. Mine some more blocks:

```
# miner.start()
```

Wait for some blocks to be mined in "docker-compose up" terminal and then stop mining.

```
# miner.stop()
```

25. Check the balance for receiver "hsm://geth1/m/44'/60'/0'/1/1" and it must be 500000000000000000.

```
# eth.getBalance("0x3f1947217ee231e02d203a81ebeee73db3248435")
```

```
> eth.getBalance("0x3f1947217ee231e02d203a81ebeee73db3248435")
500000000000000000
>
```

This completes the integration of Ethereum Blockchain with Luna HSM using BIP32 keys. If you close the HSM wallet, you will not able to perform any transaction till the wallet is opened again and account keys from Luna HSM are available for signing the transaction.

Integrate Ethereum with Luna HSM using ECDSA keys

1. Ensure that the go executable is in the PATH.

```
# export PATH=/usr/local/go/bin:$PATH
```

2. Clone the Go-Ethereum git repository on the client

```
# git clone https://github.com/ThalesGroup/go-ethereum
```

3. Change directory to the cloned repo.

```
# cd go-ethereum
```

4. Build the Geth binary.

```
# make geth
```

5. Build a Docker image for a Geth node.

```
# cd example
# make build
```

6. Ensure that Docker images are built successfully.

```
# docker images
```

```
[root@lkb-ms example]# docker images
REPOSITORY    TAG       IMAGE ID       CREATED          SIZE
geth          latest    aa20b3b73113   7 seconds ago    330MB
ubuntu        xenial    b6f507652425   14 months ago    135MB
[root@lkb-ms example]#
```

7. Create a Geth config file in the go-ethereum/example directory.

```
# cd /opt/go-ethereum/example
# ../build/bin/geth --networkid 8000 dumpconfig > config.toml
```

```
[root@lkb-ms lunaclient]# cd /opt/go-ethereum/example
[root@lkb-ms example]# ../build/bin/geth --networkid 8000 dumpconfig > config.toml
INFO [11-15|18:58:01.906] Maximum peer count                       ETH=50 LES=0 total=50
INFO [11-15|18:58:01.908] Smartcard socket not found, disabling    err="stat /run/pcscd/pcscd.comm: no such file or directory"
INFO [11-15|18:58:01.913] Set global gas cap                       cap=50,000,000
[root@lkb-ms example]#
```

<u>**For Luna Cloud HSM:**</u>

Create two geth config files in go-ethereum/example directory as we have two different service client.

```
#  ../build/bin/geth --networkid 8000 dumpconfig > config1.toml
#  ../build/bin/geth --networkid 8000 dumpconfig > config2.toml
```

8. Add the following configuration to the [Node] section in config.toml.

```
NoPKCS11BIP32 = true
PKCS11Lib = "/usr/local/lunaclient/libs/64/libCryptoki2.so"
```

```
[Node]
NoPKCS11BIP32 = true
PKCS11Lib = "/usr/local/lunaclient/libs/64/libCryptoki.so"
DataDir = "/root/.ethereum"
IPCPath = "geth.ipc"
```

<u>**For Luna Cloud HSM:**</u>

In config1.toml:

```
NoPKCS11BIP32 = true
PKCS11Lib = "/etc/geth/lunaconf1/libs/64/libCryptoki2.so"
```

In config2.toml:

```
NoPKCS11BIP32 = true
PKCS11Lib = "/etc/geth/lunaconf2/libs/64/libCryptoki2.so"
```

9. Create two ethash directories for the DAG in go-ethereum/example directory.

```
# mkdir ethash1
# mkdir ethash2
```

10. Initialize the data directories:

```
# make clean
# make init
```

11. If you are using Luna Cloud HSM, update the docker-compose.yaml under go-ethereum/example directory. Make the following changes:

> **Note:** If using Luna HSM, Skip this step as no update is required for Luna HSM. These modification in docker-compose.yaml is required only for Luna Cloud HSM.

a. Remove the following line from [volumes] section of geth1.

```
- /usr/local/lunaclient:/usr/local/lunaclient
```

b. Make the following change in the volume section of geth1:

Update:

```
./config.toml:/etc/geth/config.toml
```

To

```
./config1.toml:/etc/geth/config.toml
```

c. Remove the following line from [volumes] section of geth2.

```
- /usr/local/lunaclient:/usr/local/lunaclient
```

d. Make the following change in the volume section of geth2:

Update:

```
./config.toml:/etc/geth/config.toml
```

To

```
./config2.toml:/etc/geth/config.toml
```

Ensure that the docker-compose.yaml file contains the following details:

```
geth1:
  container_name: geth1
  image: geth
  working_dir: "/"
  environment:
    - ChrystokiConfigurationPath=/etc/geth/lunaconf1
  volumes:
    - /etc/geth/lunaconf1:/etc/geth/lunaconf1
    - ./config1.toml:/etc/geth/config.toml
    - ./data1:/etc/geth/data
    - ./ethash1:/etc/geth/ethash
  ports:
    - "30303"
  networks:
    vpcbr:
      ipv4_address: 10.8.0.3


geth2:
  container_name: geth2
  image: geth
  working_dir: "/"
  environment:
    - ChrystokiConfigurationPath=/etc/geth/lunaconf2
  volumes:
    - /etc/geth/lunaconf2:/etc/geth/lunaconf2
    - ./config2.toml:/etc/geth/config.toml
    - ./data2:/etc/geth/data
    - ./ethash2:/etc/geth/ethash
  ports:
    - "30303"
  networks:
    vpcbr:
      ipv4_address: 10.8.0.4
```

12. Create two Docker containers (geth1 and geth2) running geth image.

```
# docker-compose up
```

13. In two separate terminals, attach to geth1 and geth2 nodes.

```
# ./console.sh <node>
```

In terminal 1:

```
# ./console.sh 1
```

In terminal 2:

```
# ./console.sh 2
```

14. In both consoles (geth1, geth2), open the wallet. Provide the partition password when prompted.

In terminal 1:

```
# personal.openWallet("hsm://<partition1>")
```

```
> personal.openWallet("hsm://TPA01")
Please enter the partition password:
null
>
```

In terminal 2:

```
# personal.openWallet("hsm://<partition2>")
```

```
> personal.openWallet("hsm://TPA02")
Please enter the partition password:
null
>
```

15. Check the status of wallet in both terminals.

```
# personal.listWallets
```

Terminal 1:

```
> personal.listWallets
[{
    status: "Open",
    url: "hsm://TPA01"
}]
>
```

Terminal 2:

```
> personal.listWallets
[{
    status: "Open",
    url: "hsm://TPA02"
}]
>
```

16. In both consoles (geth1, geth2), create an account.

    In terminal 1:

    ```
    # personal.newHsmAccount("hsm://<partition1>")
    ```

    ```
    > personal.newHsmAccount("hsm://TPA01")
    {
      address: "0x8babb7fbb843d3300faf22ef61db4520982ae3e0",
      url: "hsm://TPA01/0x8babb7fbb843d3300faf22ef61db4520982ae3e0"
    }
    ```

    In terminal 2:

    ```
    # personal.newHsmAccount("hsm://<partition2>")
    ```

    ```
    > personal.newHsmAccount("hsm://TPA02")
    {
      address: "0x13fb9c7967edf15759be769f143de81187b5cbef",
      url: "hsm://TPA02/0x13fb9c7967edf15759be769f143de81187b5cbef"
    }
    ```

    NOTE: Accounts can also be created outside of the geth node:

    geth --config config.toml account newhsm <partition>

    This will generate an ECDSA key pair on Luna HSM partition.

17. Find the node address of the geth2 account in terminal2 (geth2 console).

    ```
    # admin.nodeInfo
    ```

    The output displays the enode field containing node address.

    ```
    > admin.nodeInfo
    {
      enode: "enode://b5f7e23f47277e34c8d7cefe066473e0b522eecce5407f2a5f13a6c794ae9964bf0dc5805a0cd4a8d93abd390d1535c1a7bea9e914f0c63f6c57e715aaf84910@127.0.0.1:30304",
      enr: "enr:-KO4QOulehMgSX3AN0v8X0Knv_C2kXqmvtEfyLAy42RPIKmQclAd3WeFpBMzYC2PYchFhgWf90DGr9FH9SGpCRTgExGGAYR7f2eHg2V0aMfGhBHUfCyAgmlkgnY0gmlwhH8AAAGJc2VjcDI1NmsxoQK19-I_Ryd-NMjXzv4GZHPgtSLuz
    OVAfypfE6bHlK6ZZIRzbmFwwIN0Y3CCdmCDdWRwgnZg",
      id: "ba21eb450ddadb182f4050da5f2e67d952b3994aa0229ca4296228503269d2ca",
      ip: "127.0.0.1",
      listenAddr: "[::]:30304",
      name: "Geth/v1.10.26-stable-2460cf56/linux-amd64/go1.19.3",
      ports: {
        discovery: 30304,
        listener: 30304
      },
      protocols: {
        eth: {
          config: {
            chainId: 876,
            eip150Block: 0,
            eip150Hash: "0x0000000000000000000000000000000000000000000000000000000000000000",
            eip155Block: 0,
            eip158Block: 0,
            homesteadBlock: 0
          },
          difficulty: 1024,
          genesis: "0xd1a12dd8ee31c1b49425acee37da3070510d0121922250908ce381ac8a4c8725",
          head: "0xd1a12dd8ee31c1b49425acee37da3070510d0121922250908ce381ac8a4c8725",
          network: 8000
        },
        snap: {}
      }
    }
    ```

18. Add geth2 as peer in terminal1(geth1 console) by mentioning the enode address obtained in previous step through replacing the IP of geth2 container to 10.8.0.4.

```
# admin.addPeer("<geth2-enode-address>")
```

For example:

```
#admin.addPeer("enode://b5f7e23f47277e34c8d7cefe066473e0b522eecce5407f2a5f13a6c794ae9964bf0dc5805a0cd4a8d93abd390d1535c1a7bea9e914f0c63f6c57e715aaf84910@10.8.0.4:30304")
```

```
> admin.addPeer("enode://b5f7e23f47277e34c8d7cefe066473e0b522eecce5407f2a5f13a6c794ae9964bf0dc5805a0cd4a8d93abd390d1535c1a7bea9e914f0c63f6c57e715aaf84910@10.8.0.4:30304")
true
>
```

19. Confirm that each node assures the other as peers by running the following command in each terminal.

```
# admin.peers
```

```
> admin.peers
[{
    caps: ["eth/66", "eth/67", "snap/1"],
    enode: "enode://b5f7e23f47277e34c8d7cefe066473e0b522eecce5407f2a5f13a6c794ae9964bf0dc5805a0cd4a8d93abd390d1535c1a7bea9e914f0c63f6c57e715aaf84910@10.8.0.4:30304",
    id: "ba21eb450ddadb182f4050da5f2e67d952b3994aa0229ca4296228503269d2ca",
    name: "Geth/v1.10.26-stable-2460cf56/linux-amd64/go1.19.3",
    network: {
      inbound: false,
      localAddress: "10.8.0.3:40780",
      remoteAddress: "10.8.0.4:30304",
      static: true,
      trusted: false
    },
    protocols: {
      eth: {
        difficulty: 1024,
        head: "0xd1a12dd8ee31c1b49425acee37da3070510d0121922250908ce381ac8a4c8725",
        version: 67
      },
      snap: {
        version: 1
      }
    }
}]
```

Once both the nodes have been added as peers, you can run miner.start() to start mining. You need at least one miner to mine for any transactions to be included on the blockchain. Initially, the blockchain will build the Directed Acyclic Graph (DAG) which may take a few minutes. It will display Generating DAG in progress. After mining commences, you can see the mined test ether in the account.

20. Start the miner in terminal1 (geth1 console).

```
# miner.start()
```

```
> miner.start()
null
```

Wait for the completion of DAG generation process. The activity can be seen in the "docker-compose up" terminal. This can take some time to complete.

```
geth1    | INFO [11-17|13:59:15.439] Generating DAG in progress               epoch=0 percentage=94 elapsed=3m17.913s
geth1    | INFO [11-17|13:59:17.523] Generating DAG in progress               epoch=0 percentage=95 elapsed=3m19.997s
geth1    | INFO [11-17|13:59:19.591] Generating DAG in progress               epoch=0 percentage=96 elapsed=3m22.065s
geth1    | INFO [11-17|13:59:21.745] Generating DAG in progress               epoch=0 percentage=97 elapsed=3m24.219s
geth1    | INFO [11-17|13:59:23.940] Generating DAG in progress               epoch=0 percentage=98 elapsed=3m26.414s
geth1    | INFO [11-17|13:59:26.162] Generating DAG in progress               epoch=0 percentage=99 elapsed=3m28.636s
```

You will notice several blocks in the “docker-compose up” terminal similar to the following after the DAG generation process is completed.

```
geth1     | INFO [11-15|14:27:03.288] 🔨 mined potential block                number=3 hash=39c7e5..6d334b
geth2     | WARN [11-15|14:27:03.292] Snap syncing, discarded propagated block number=3 hash=39c7e5..6d334b
geth1     | INFO [11-15|14:27:03.292] Commit new sealing work                 number=4 sealhash=0de010..2ed7a6 uncles=0 txs=0 gas=0 fees=0 elapsed=4.346ms
geth1     | INFO [11-15|14:27:03.292] Commit new sealing work                 number=4 sealhash=0de010..2ed7a6 uncles=0 txs=0 gas=0 fees=0 elapsed=4.458ms
geth1     | INFO [11-15|14:27:04.777] Generating DAG in progress              epoch=1 percentage=0  elapsed=7.892s
geth1     | INFO [11-15|14:27:04.946] Successfully sealed new block           number=4 sealhash=0de010..2ed7a6 hash=39e425..375c4b elapsed=1.653s
geth1     | INFO [11-15|14:27:04.946] 🔨 mined potential block                number=4 hash=39e425..375c4b
geth1     | INFO [11-15|14:27:04.960] Commit new sealing work                 number=5 sealhash=443633..195686 uncles=0 txs=0 gas=0 fees=0 elapsed="153.459µs"
geth1     | INFO [11-15|14:27:04.960] Commit new sealing work                 number=5 sealhash=443633..195686 uncles=0 txs=0 gas=0 fees=0 elapsed="243.902µs"
geth1     | INFO [11-15|14:27:06.043] Successfully sealed new block           number=5 sealhash=443633..195686 hash=f04838..37479d elapsed=1.082s
```

21. Stop the miner in terminal1 (geth1 console) after the DAG generation process gets completed and after some blocks have been mined,.

    ```
    # miner.stop()
    ```

22. Find the geth1 account address by executing the following command in terminal1 (geth1 console).

    ```
    # personal.listAccounts
    ```

    ```
    > personal.listAccounts
    ["0x8babb7fbb843d3300faf22ef61db4520982ae3e0"]
    >
    ```

23. Get the balance of the geth1 account in either terminal 1 or terminal 2. The account will show some ether in when queried.

    ```
    # eth.getBalance("<geth1-account-address>")
    ```

    ```
    > eth.getBalance("0x8babb7fbb843d3300faf22ef61db4520982ae3e0")
    130000000000000000000
    >
    ```

24. Perform a transaction from the geth1 account to the geth2 account. Find the geth1 account address by executing `personal.listAccounts` in terminal1 (geth1 console) and use it as sender account address. Find the geth2 account address by executing `personal.listAccounts` in terminal2 (geth2 console) and use it as receiver account address to transfer ethers.

    In terminal 1:

    ```
    # eth.sendTransaction({from:sender, to:receiver, value: amount})
    ```

    For example:

    ```
    # eth.sendTransaction({from: "0x8babb7fbb843d3300faf22ef61db4520982ae3e0", to:
    "0x13fb9c7967edf15759be769f143de81187b5cbef", value: web3.toWei(0.5, "ether")})
    ```

    ```
    > eth.sendTransaction({from: "0x8babb7fbb843d3300faf22ef61db4520982ae3e0", to: "0x13fb9c7967edf15759be769f143de81187b5cbef", value: web3.toWei(0.5, "ether")})
    "0xd46f5b91ab5cfe3dfe9303dcc4667bc581f3a1498d4c6c6978b4e3ef2af94809"
    >
    ```

25. You can view the transcription by executing the following command on terminal 1:

```
# eth.pendingTransactions
```

```
> eth.pendingTransactions
[{
    blockHash: null,
    blockNumber: null,
    chainId: "0x36c",
    from: "0x8babb7fbb843d3300faf22ef61db4520982ae3e0",
    gas: 21000,
    gasPrice: 1000000000,
    hash: "0xd46f5b91ab5cfe3dfe9303dcc4667bc581f3a1498d4c6c6978b4e3ef2af94809",
    input: "0x",
    nonce: 0,
    r: "0x667dc93cd512ea2b1d2810d4020a3e2a14c90aef9b0135f51aca4aa7861c0dc",
    s: "0x3f56ff61068d561237645a100656cd6594fa43c0faf5e751353479331c4c360f",
    to: "0x13fb9c7967edf15759be769f143de81187b5cbef",
    transactionIndex: null,
    type: "0x0",
    v: "0x6fc",
    value: 500000000000000000
}]
```

26. Mine some more blocks in terminal 1.

```
# miner.start()
```

27. Wait for some blocks to be mined in "docker-compose up" terminal and then stop mining.

```
# miner.stop()
```

28. In any terminal, observe the balance of the geth2 account.

```
# eth.getBalance("<geth2-account-address>")
```

Observe the balance transferred in the transaction performed. As per the transaction, the balance in geth2 account should be: 500000000000000000

```
> eth.getBalance("0x13fb9c7967edf15759be769f143de81187b5cbef")
500000000000000000
>
```

29. Close the wallets.

In terminal 1:

```
# personal.closeWallet("hsm://<partition1>")
```

In terminal 2:

```
# personal.closeWallet("hsm://<partition2>")
```

After closing the wallet, you cannot perform any transaction as the signing keys from HSM is not available.

This completes the integration of Go-Ethereum Blockchain with Luna HSM. It generates the ECDSA signing keys in the HSM partition that are then used by the Ethereum accounts for signing blockchain transactions.

Contacting Customer Support

If you encounter a problem while installing, registering, or operating this product, refer to the documentation. If you cannot resolve the issue, contact your supplier or Thales Customer Support. Thales Customer Support operates 24 hours a day, 7 days a week. Your level of access to this service is governed by the support plan arrangements made between Thales and your organization. Please consult this support plan for further information about your entitlements, including the hours when telephone support is available to you.

Customer Support Portal

The Customer Support Portal, at https://supportportal.thalesgroup.com, is a database where you can find solutions for most common problems. The Customer Support Portal is a comprehensive, fully searchable repository of support resources, including software and firmware downloads, release notes listing known problems and workarounds, a knowledge base, FAQs, product documentation, technical notes, and more. You can also use the portal to create and manage support cases.

> **NOTE:** You require an account to access the Customer Support Portal. To create a new account, go to the portal and click on the **REGISTER** link.

Telephone Support

If you have an urgent problem, or cannot access the Customer Support Portal, you can contact Thales Customer Support by telephone at +1 410-931-7520. Additional local telephone support numbers are listed on the support portal.