Ethereum Blockchain

SafeNet Luna HSM Integration Guide

gemalto
security to be free

Document Part Number: 007-000189-001, Rev. A

Release Date: December 2018

Contents

Preface

This document is intended to guide administrators through the steps of supporting Ethereum Blockchain technology with SafeNet HSMs including configuration and integration.

Scope

This guide is intended to show SafeNet HSMs protecting the private keys used to sign Ethereum Blockchain transactions for Ethereum accounts.

Document Conventions

This section provides information on the conventions used in this template.

Notes

Notes are used to alert you to important or helpful information. These elements use the following format:

NOTE: Take note. Contains important or helpful information.

Cautions

Cautions are used to alert you to important information that may help prevent unexpected results or data loss. These elements use the following format:

CAUTION: Exercise caution. Caution alerts contain important information that may help prevent unexpected results or data loss.

Warnings

Warnings are used to alert you to the potential for catastrophic data loss or personal injury. These elements use the following format:

WARNING: Be extremely careful and obey all safety and security measures. In this situation you might do something that could result in catastrophic data loss or personal injury.

Command Syntax and Typeface Conventions

Convention	Description
bold	The bold attribute is used to indicate the following: • Command-line commands and options (Type **dir /p**.) • Button names (Click **Save As**.) • Check box and radio button names (Select the **Print Duplex** check box.) • Window titles (On the **Protect Document** window, click **Yes**.) • Field names (**User Name:** Enter the name of the user.) • Menu names (On the **File** menu, click **Save**.) (Click **Menu** > **Go To** > **Folders**.) • User input (In the **Date** box, type **April 1**.)
italic	The italic attribute is used for emphasis or to indicate a related document. (See the *Installation Guide* for more information.)
`Consolas`	Denotes syntax, prompts, and code examples.

Support Contacts

Contact Method	Contact Information	
Address	Gemalto 4690 Millennium Drive Belcamp, Maryland 21017, USA	
Phone	US	1-800-545-6608
	International	1-410-931-7520
Technical Support Customer Portal	https://supportportal.gemalto.com Existing customers with a Technical Support Customer Portal account can log in to manage incidents, get the latest software upgrades, and access the Gemalto Knowledge Base.	

1
Introduction

Overview

This guide demonstrates using a SafeNet Luna HSM or HSM on Demand (HSMoD) 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 using the SafeNet HSM. You then use the SafeNet HSM to sign transactions. The HSM Wallet uses a PKCS#11 API to communicate with the HSM. Each PKCS#11 partition corresponds to a different HSM Wallet.

The benefits of using SafeNet HSMs to generate the signing keys for Ethereum Blockchain Accounts include the following:

- Secure generation, storage and protection of the signing private key on FIPS 140-2 level 3 validated hardware*.
- Full life cycle management of the keys.
- HSM audit trail**.
- Take advantage of cloud services with confidence.
- Significant performance improvements by off-loading cryptographic operations from application servers.

*validation for HSMoD services in progress.
**HSMoD services do not have access to the secure audit trail.

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 a 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**.

Refer *Go-Ethereum documentation* for more information on its implementation and working.

Third Party Application Details

This integration guide uses the following third party applications:

- Go-Ethereum

Supported Platforms

Below is the list of the platforms tested with the following HSMs:

SafeNet Luna HSM: SafeNet 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. SafeNet Luna HSMs physically and logically secure cryptographic keys and accelerate cryptographic processing.

The SafeNet Luna HSM on premise offerings include the SafeNet Luna Network HSM, SafeNet PCIe HSM, and SafeNet Luna USB HSMs. SafeNet Luna HSMs are also available for access as an offering from cloud service providers such as IBM cloud HSM and AWS cloud HSM classic.

Platforms tested:

- RHEL
- Ubuntu

NOTE: Follow *HSM product documentation* for HSM version supporting BIP32 mechanism.

SafeNet DPoD: is a cloud-based platform that provides on-demand HSM and Key Management services through a simple graphical user interface. With DPoD, 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 just the services you need.

Platforms tested:

- RHEL
- Ubuntu

NOTE: BIP32 mechanism is not supported with DPoD as of now.

Prerequisites

Before you proceed with the integration, ensure that you have completed the following.

Configure SafeNet HSM

The SafeNet Luna HSM or HSM on Demand Service configuration procedural material is included in the relevant chapter of this integration guide.

Install Ethereum Blockchain

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

To install the prerequisite libraries

Install the following components on the system.

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

To install and setup Golang

Ethereum Blockchain uses the Go programming language for many of its components. Install the **golang** using the following URL.

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

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

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

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

To install and setup 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/install/linux/docker-ee/rhel/

Install the **docker-compose:**

```
# sudo pip install docker-compose==<version>
```

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

To configure Go-Ethereum

Clone the Go-Ethereum git and prepare the Docker library.

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

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

2. Change directory to the cloned repo.

   ```
   # cd go-ethereum
   ```

3. Build the geth binary.

```
# make geth
```

4. Build a docker image for a geth node.

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

```
[root@localhost example]# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
geth                latest              74458e030210        16 minutes ago      300MB
ubuntu              xenial              7aa3602ab41e        4 weeks ago         115MB
```

2

Integrating Ethereum Blockchain with HSM on Demand service

Configuring Ethereum Blockchain using SafeNet HSMoD

SafeNet Data Protection on Demand (DPoD) provides strong cloud protection of secure assets, including keys, and should be considered a best practice when building systems based on Ethereum Blockchain.

Provisioning your HSM on Demand service

This service provides your client machine with access to an HSM Application Partition for storing cryptographic objects used by your applications. Application partitions can be assigned to a single client, or multiple clients can be assigned to, and share, a single application partition.

NOTE: Refer the “SafeNet Data Protection on Demand Application Owner Quick Start Guide” for more information about provisioning the HSM on Demand service and initializing the service client.

The HSM on Demand service client package is a zip file that contains system information you require to connect your client machine to an existing HSM on Demand service.

To create the Ethereum Key Vault services in Data Protection on Demand

1. Create the following Key Vault services in Data Protection on Demand
   - `geth1`
   - `geth2`
2. For each service, create a client and download the zip to the host system.
3. Make the following directories on the client machine:

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

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

4. Unzip the client zip files for **geth1** and **geth2** in to the directories **lunaconf1** and **lunaconf2** respectively.
5. Follow the instructions in the *DPOD Application Owner Quick Start Guid*e and initialize both the partitions, Security Officer, Crypto Officer and Crypto User roles.

6. Verify that both partitions are visible from the client via the `lunacm` utility.

```
[root@localhost 64]# cd /etc/geth/lunaconf1
[root@localhost lunaconf1]# source ./setenv
[root@localhost lunaconf1]# ./bin/64/lunacm
LunaCM v1.0.0-638. Copyright (c) 2006-2017 SafeNet.

        Available HSMs:

        Slot Id ->              3
        Label ->                geth1
        Serial Number ->        1294607986107
        Model ->                Luna K7
        Firmware Version ->     7.1.1
        Configuration ->        Luna User Partition With SO (PW) Signing With Cloning Mode
        Slot Description ->     User Token Slot


        Current Slot Id: 3

lunacm:>e
[root@localhost lunaconf1]# cd /etc/geth/lunaconf2
[root@localhost lunaconf2]# source ./setenv
[root@localhost lunaconf2]# ./bin/64/lunacm
LunaCM v1.0.0-638. Copyright (c) 2006-2017 SafeNet.

        Available HSMs:

        Slot Id ->              3
        Label ->                geth2
        Serial Number ->        1294607986108
        Model ->                Luna K7
        Firmware Version ->     7.1.1
        Configuration ->        Luna User Partition With SO (PW) Signing With Cloning Mode
        Slot Description ->     User Token Slot


        Current Slot Id: 3

lunacm:>e
```

Integrating Ethereum with SafeNet DPoD

This section provides instructions on configuring Go-Ethereum with SafeNet DPoD using ECDSA keys.

To integrate Ethereum with SafeNet DPoD using ECDSA keys

1. Create two **geth** config files in `go-ethereum/example` directory.

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

2. Add the following configuration to the `[Node]` section in the corresponding configuration files:

   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"
   ```

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

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

4. Initialize the data directories:

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

5. Edit the docker-compose.yaml under go-ethereum/example directory. Make the following changes:
   a. Remove the line - /usr/local/lunaclient:/usr/local/lunaclient under [volumes] section of geth1 and geth2
   b. Under `geth1:volumes`, edit

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

      to

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

      Under `geth2:volumes`, edit

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

      to

      `./config2.toml:/etc/geth/config.toml`
6. Create two docker containers (geth1 and geth2) by running geth image.

   ```
   # docker-compose up
   ```
7. 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`
8. In both consoles (geth1, geth2), open the HSM wallet. Provide the partition password when prompted.

   In terminal 1:

   ```
   # personal.listWallets
   # personal.openWallet("hsm://geth1")
   ```

   In terminal 2:

   ```
   # personal.listWallets
   # personal.openWallet("hsm://geth2")
   ```
9. In both consoles (geth1, geth2), create an account.

   In terminal 1:

   ```
   # personal.newHsmAccount("hsm://geth1")
   ```

   In terminal 2:

   ```
   # personal.newHsmAccount("hsm://geth2")
   ```

   This will generate ECDSA key pair on both partitions.

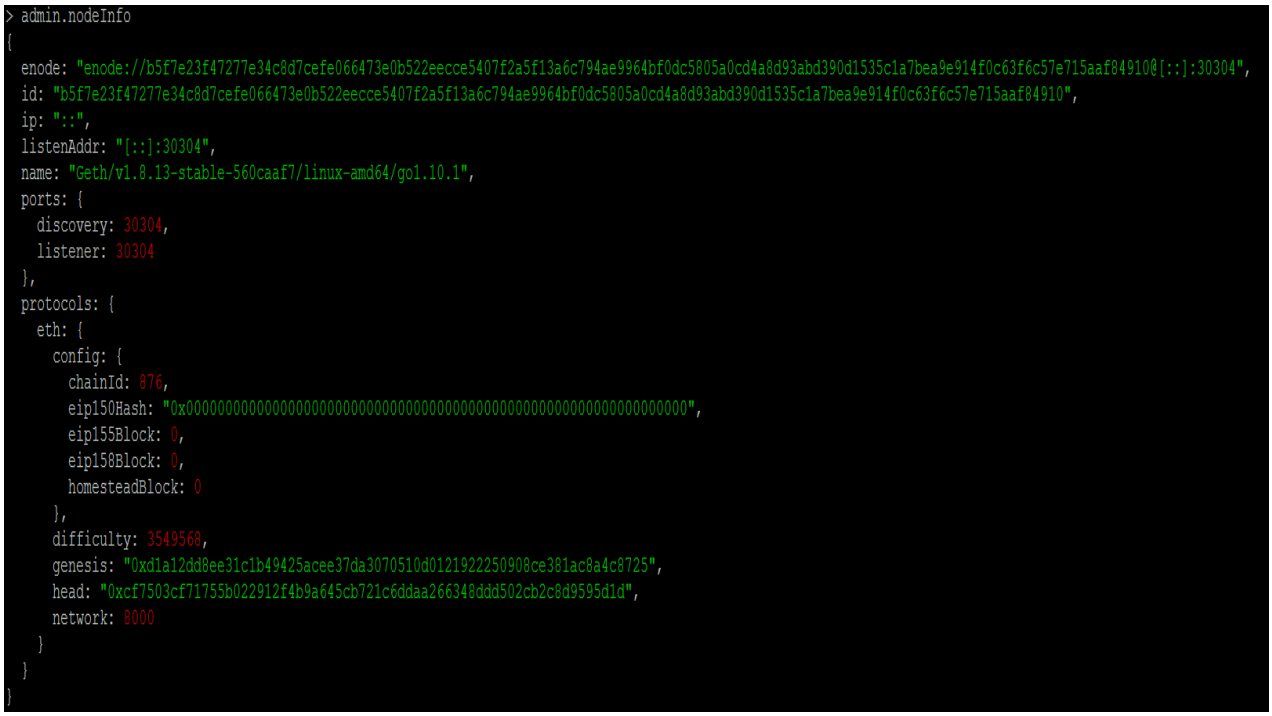
NOTE: Accounts can also be created outside of the geth node:

```
geth --config config.toml account newhsm geth1
```

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

    ```
    # admin.nodeInfo
    ```

    The output contains the enode field:



11. Add geth2 as peer in terminal1(geth1 console) by mentioning the enode address obtained in above step and replacing [::] to 10.8.0.4

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

    For Example:

    ```
    #admin.addPeer("enode://b5f7e23f47277e34c8d7cefe066473e0b522eecce5407f2a5f13a6c794ae9964bf0dc580
    5a0cd4a8d93abd390d1535c1a7bea9e914f0c63f6c57e715aaf84910@10.8.0.4:30304")
    ```

12. Confirm that each node sees each other as peers by running the following in each terminal:

    ```
    # admin.peers
    ```

```
> admin.peers
[{
    caps: ["eth/63"],
    id: "b5f7e23f47277e34c8d7cefe066473e0b522eecce5407f2a5f13a6c794ae9964bf0dc58
05a0cd4a8d93abd390d1535c1a7bea9e914f0c63f6c57e715aaf84910",
    name: "Geth/v1.8.14-unstable-08634462/linux-amd64/go1.10.1",
    network: {
      inbound: false,
      localAddress: "10.8.0.3:50808",
      remoteAddress: "10.8.0.4:30304",
      static: true,
      trusted: false
    },
    protocols: {
      eth: {
        difficulty: 1024,
        head: "0xd1a12dd8ee31c1b49425acee37da3070510d0121922250908ce381ac8a4c872
5",
        version: 63
      }
    }
}]
```

Performing Transactions

Once the setup is done, you can run miner.start() to start mining—you need at least one miner to mine in order for any transactions to be included on the Blockchain. Initially it will build the Directed Acyclic Graph (DAG) which may take a few minutes. It displays **Generating DAG** in progress.

After mining commences, you can see the mined test ether in the account.

To see the mined test ether in the account

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

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

   Wait for "Generating DAG in progress" to complete. The activity can be seen in the "docker-compose up" terminal. This can take some time to complete.

2. When completed and after some blocks have been mined, stop the miner.

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

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

   ```
   # personal.listAccounts
   ```

```
> personal.listAccounts
["0x474bca912e9f2fbbd3a0636402d876f87a0ec0cd"]
```

4. Get the balance of the geth1 account in terminal 1/terminal 2. The account should display some ether in either terminal.

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

   For Example:

```
> eth.getBalance("0x474bca912e9f2fbbd3a0636402d876f87a0ec0cd")
3.4245e+21
>
```

5. Send 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.

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

   For Example:

   ```
   # eth.sendTransaction({from: "0x474bca912e9f2fbbd3a0636402d876f87a0ec0cd", to:
   "0xd5feed8d1a1141a27f9545b32bf52e3f9316de53", value: web3.toWei(0.5, "ether")})
   ```

   You can view the transaction by executing the following command on terminal 1:

   ```
   # eth.pendingTransactions
   ```

   ```
   > eth.pendingTransactions
   [{
       blockHash: null,
       blockNumber: null,
       from: "0x474bca912e9f2fbbd3a0636402d876f87a0ec0cd",
       gas: 90000,
       gasPrice: 18000000000,
       hash: "0x0aeb8d18f3856da20b2f8a35e69c931cb33b2d2f5ceda390140a320e4ca2dba9",
       input: "0x",
       nonce: 0,
       r: "0xbe9bcc194ea402a4280ddde6f47e5a2d2c0dd8bfdd82d1982d81de0c1c9b0cea",
       s: "0x137c50beec4ceb70b5289036006f1adf2cfd0e461bde43e561e68a71a2aa5819",
       to: "0xd5feed8d1a1141a27f9545b32bf52e3f9316de53",
       transactionIndex: 0,
       v: "0x6fc",
       value: 500000000000000000
   }]
   ```

6. Mine some more blocks in terminal 1.

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

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

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

7. In both consoles, observe the balance of the geth2 account.

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

   You should observe the balance transferred in the transaction.

   In the above example, the balance in geth2 account should be: 500000000000000000

   ```
   > eth.getBalance("0xd5feed8d1a1141a27f9545b32bf52e3f9316de53")
   500000000000000000
   ```

8. Close the geth1 and geth2 wallets.

   In terminal 1:

   ```
   # personal.closeWallet("hsm://geth1")
   ```

   In terminal 2:

   ```
   # personal.closeWallet("hsm://geth2")
   ```

This completes the integration of Go-Ethereum Blockchain with SafeNet DPoD. The HSMoD service generates the ECDSA signing keys that are then used by the Ethereum accounts in their Blockchain transaction.

3

Integrating Ethereum Blockchain with SafeNet Luna HSM

Configuring Ethereum Blockchain using SafeNet Luna HSM

SafeNet HSM provides strong protection of secure assets, including keys, and should be considered a best practice when building systems based on Ethereum Blockchain.

Configuring the SafeNet Luna HSM

Before you get started ensure the following:

1. Ensure the HSM is setup, initialized, provisioned and ready for deployment. Refer to the HSM product documentation for help.
2. Create two partitions on the HSM geth1 and geth2 that will be later used by Ethereum blockchain.
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. Install a full Luna HSM Client software (non-minimal) on the client machine (default installation directory: `usr/safenet/lunaclient`).
5. Extract the Luna Client minimum installer in folder /usr/local/lunaclient.

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

6. Set the ChrystokiConfigurationPath variable to point to the directory /etc/geth/lunaconf1.
7. Copy the Chrystoki-template.conf to the /etc/geth/lunaconf1 directory.

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

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

   ```
   Chrystoki2 = {

   LibUNIX64 = /usr/local/lunaclient/libs/64/libCryptoki2.so;

   }
   ```

NOTE: To enable the cklog, add `Cklog2` section configuration in `Chrystoki.conf`.

9. Modify the LunaSA Client section of /etc/geth/lunaconf1/Chrystoki.conf to:

```
ClientPrivKeyFile = /etc/geth/lunaconf1/;
ClientCertFile = /etc/geth/lunaconf1/;
ServerCAFile = /etc/geth/lunaconf1/CAFile.pem;
```

10. Create client certificate :

```
# /usr/safenet/lunaclient/bin/vtl createCert -n geth1
```

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

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

12. Add the server :

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

13. Transfer the client certificate to the HSM.

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

14. Login to HSM and register the client and assign geth1 partition to the client.
15. From client machine, use lunacm utility to initialize the partition, and Crypto Officer role.
16. Repeat steps #6 to #15 for client/partition2 replacing lunaconf1 and geth1 with lunaconf2 and geth2 respectively.

Integrating Ethereum with SafeNet Luna HSM

This section provides instructions on configuring Go-ethereum with SafeNet Luna HSM.

- To integrate Ethereum with SafeNet Luna HSM using BIP32 keys
- To integrate Ethereum with SafeNet Luna HSM using ECDSA keys

To integrate Ethereum with SafeNet Luna HSM using BIP32 keys

1. Create a geth config file in `go-ethereum/example` directory

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

2. Add the following configuration to the `[Node]` section in `config.toml`:

```
NoPKCS11BIP32 = false
PKCS11Lib = "/usr/local/lunaclient/libs/64/libCryptoki.so"
```

NOTE: To enable the cklog, specify the *libcklog2.so* instead of *libCryptoki.so*.

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

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

4. Initialize the data directories:

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

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

```
# docker-compose up
```

5. Attach to node 1:

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

NOTE: If you have enabled cklog, you can see the cklog output in separate terminal by executing :

```
./cklog.sh <node>
```

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

```
# personal.listWallets
```

```
> personal.listWallets
[{
    status: "Closed",
    url: "hsm://geth1"
}]
```

7. In console 1 open the wallet. Provide the partition password when prompted.

```
# personal.openWallet("hsm://geth1")
```

It will prompt for partition password. On executing this first time, it will generate master seed and derive a master key pair.

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

8. Derive an account:

```
# personal.deriveAccount("hsm://geth1", "m/44'/60'/0'/1/0", true)
```

```
> personal.deriveAccount("hsm://geth1", "m/44'/60'/0'/1/0", true)
{
  address: "0xc9b4e881ea4ce40e0f284a82116f8be28d74d6ae",
  url: "hsm://geth1/m/44'/60'/0'/1/0"
}
```

The "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”*

9. Set the coinbase to the new derived account:

```
# miner.setEtherbase("<derive account address>")
```
where derive account is obtained from the above step.

For Example :
```
# miner.setEtherbase("0xc9b4e881ea4ce40e0f284a82116f8be28d74d6ae")
```

10. Fund the account by starting the miner:
```
# miner.start()
```
Wait for "`Generating DAG in progress`" to complete. The activity can be seen in the "docker-compose up" terminal. This can take some time to complete.

11. Check the balance using :
```
# eth.getBalance("<derive account address>")
```
For Example :
```
# eth.getBalance("0xc9b4e881ea4ce40e0f284a82116f8be28d74d6ae")
```
You should be able to see some ether in this account.

```
> eth.getBalance("0xc9b4e881ea4ce40e0f284a82116f8be28d74d6ae")
65000000000000000000
```

12. When completed and after some blocks have been mined, stop the miner.
```
# miner.stop()
```

13. Derive another account:
```
# personal.deriveAccount("hsm://geth1", "m/44'/60'/0'/1/1", true)
```

```
> personal.deriveAccount("hsm://geth1", "m/44'/60'/0'/1/1", true)
{
  address: "0xdfdd354c1ec447a18f486024d1b75da04fad02e7",
  url: "hsm://geth1/m/44'/60'/0'/1/1"
}
```

14. Send funds from the first account to the second account:
```
# eth.sendTransaction({from:sender, to:receiver, value: amount})
```
For Example :
```
# eth.sendTransaction({from:"0xc9b4e881ea4ce40e0f284a82116f8be28d74d6ae",
to:"0xdfdd354c1ec447a18f486024d1b75da04fad02e7", value: web3.toWei(0.5, "ether")})
```

15. Mine some more blocks:
```
# miner.start()
```
Wait for some blocks to be mined in "docker-compose up" terminal and then stop mining.
```
# miner.stop()
```

16. Check the balance for "hsm://geth1/m/44'/60'/0'/1/1" is 500000000000000000:
```
# eth.getBalance("0xdfdd354c1ec447a18f486024d1b75da04fad02e7")
```

```
> eth.getBalance("0xdfdd354c1ec447a18f486024d1b75da04fad02e7")
500000000000000000
> miner.stop()
```

To integrate Ethereum with SafeNet Luna HSM using ECDSA keys

1. Create a geth config file in `go-ethereum/example` directory

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

2. Add the following configuration to the `[Node]` section in `config.toml`:

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

   NOTE: To enable the cklog, specify the *libcklog2.so* instead of *libCryptoki2.so*.

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

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

4. Initialize the data directories:

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

5. Create two docker containers (geth1 and geth2) running geth image.

   ```
   # docker-compose up
   ```

6. 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
   ```

   NOTE: If you have enabled cklog, you can see the cklog output in separate terminal by executing :

   ```
   ./cklog.sh <node>
   ```

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

   ```
   # personal.listWallets
   ```

   In terminal 1:

   ```
   # personal.openWallet("hsm://geth1")
   ```

   In terminal 2:

   ```
   # personal.openWallet("hsm://geth2")
   ```

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

   In terminal 1:

   ```
   # personal.newHsmAccount("hsm://geth1")
   ```

In terminal 2:

```
# personal.newHsmAccount("hsm://geth2")
```

> **NOTE:** Accounts can also be created outside of the geth node:
>
> `geth --config config.toml account newhsm geth1`

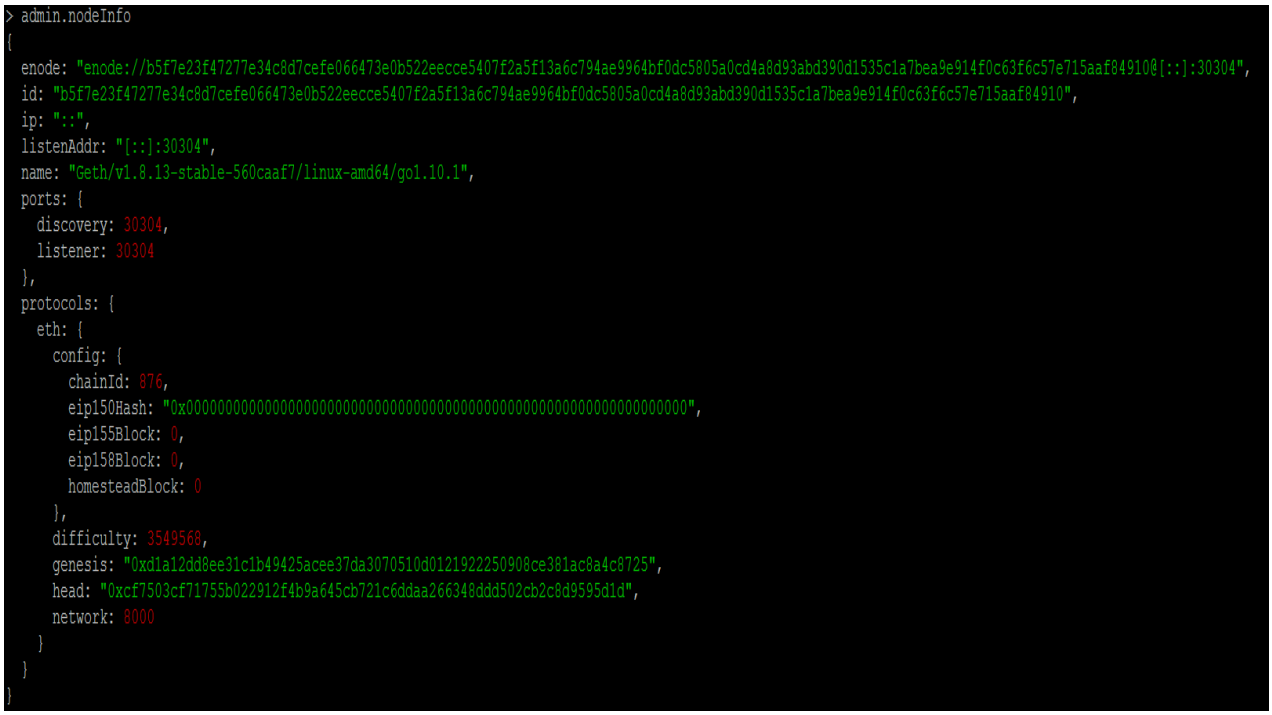
This will generate an ECDSA key pair on both partitions.

> **NOTE:** Observe the PKCS11 calls in the cklog terminal.

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

   ```
   # admin.nodeInfo
   ```

   The output displays the enode field.



10. Add geth2 as peer in terminal1(geth1 console) by mentioning the by mentioning the enode address obtained in above step and replacing [::] to 10.8.0.4

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

    For Example:

    ```
    #admin.addPeer("enode://b5f7e23f47277e34c8d7cefe066473e0b522eecce5407f2a5f13a6c794ae9964bf0dc580
    5a0cd4a8d93abd390d1535c1a7bea9e914f0c63f6c57e715aaf84910@10.8.0.4:30304")
    ```

11. Confirm that each node sees each other as peers by running the following in each terminal:

    ```
    # admin.peers
    ```

```
> admin.peers
[{
    caps: ["eth/63"],
    id: "b5f7e23f47277e34c8d7cefe066473e0b522eecce5407f2a5f13a6c794ae9964bf0dc58
05a0cd4a8d93abd390d1535c1a7bea9e914f0c63f6c57e715aaf84910",
    name: "Geth/v1.8.14-unstable-08634462/linux-amd64/go1.10.1",
    network: {
      inbound: false,
      localAddress: "10.8.0.3:50808",
      remoteAddress: "10.8.0.4:30304",
      static: true,
      trusted: false
    },
    protocols: {
      eth: {
        difficulty: 1024,
        head: "0xd1a12dd8ee31c1b49425acee37da3070510d0121922250908ce381ac8a4c872
5",
        version: 63
      }
    }
}]
```

Performing Transactions

Once the setup is done, you can run miner.start() to start mining—you need at least one miner to mine in order for any transactions to be included on the Blockchain. Initially it 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.

To see the mined test ether in the account

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

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

   Wait for "Generating DAG in progress" to complete. The activity can be seen in the "docker-compose up" terminal. This can take some time to complete.

2. When completed and after some blocks have been mined, stop the miner.

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

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

   ```
   # personal.listAccounts
   ```

```
> personal.listAccounts
["0x474bca912e9f2fbbd3a0636402d876f87a0ec0cd"]
```

4. Get the balance of the geth1 account in terminal 1/terminal 2. The account should have some ether in either terminal.

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

```
> eth.getBalance("0x474bca912e9f2fbbd3a0636402d876f87a0ec0cd")
3.4245e+21
>
```

5. Send 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.

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

   For Example:

   ```
   # eth.sendTransaction({from: "0x474bca912e9f2fbbd3a0636402d876f87a0ec0cd", to:
   "0xd5feed8d1a1141a27f9545b32bf52e3f9316de53", value: web3.toWei(0.5, "ether")})
   ```

NOTE: Observe the Sign operation in the cklog terminal.

```
16:36:06 00005-2110826240:STRTOpenSession {Slot=0 Flgs=6 }
16:36:06 00005-2110826240:FINIOpenSession CKR_OK(3428283ms) {Sesn=2 }

16:36:06 00005-2110826240:STRTFindObjects_Init {Sesn=2 AttrList={Attribute=CKA_C
LASS=(8 "0300000000000000") Attribute=CKA_LABEL=(42 "0x474bcA912E9F2FBbD3A063640
2d876F87A0eC0CD") } }
16:36:06 00005-2110826240:FINIFindObjects_Init CKR_OK(3428283ms) {}

16:36:06 00005-2110826240:STRTFindObjects {Sesn=2 UShort=1 }
16:36:06 00005-2110826240:FINIFindObjects CKR_OK(3428284ms) {ObjHndLst={Obj=158
} UShort=1 }

16:36:06 00005-2110826240:STRTSign_Init {Sesn=2 Mech=(CKM_ECDSA,(0 "<null>")) Ob
j=158 }
16:36:06 00005-2110826240:FINISign_Init CKR_OK(3428285ms) {}

16:36:06 00005-2110826240:STRTSign {Sesn=2 ByteString=(32 "6f4ad211c7218979d58eb
865d22aba315c252468232a0085283a7cca1d9c983e") pusSignatureLen=(0xc420f149a0 0) }
16:36:06 00005-2110826240:FINISign CKR_OK(3428285ms) {ByteString=(0 "<null>") }

16:36:06 00005-2110826240:STRTSign {Sesn=2 ByteString=(32 "6f4ad211c7218979d58eb
865d22aba315c252468232a0085283a7cca1d9c983e") pusSignatureLen=(0xc420f149a0 64)
}
16:36:06 00005-2110826240:FINISign CKR_OK(3428285ms) {ByteString=(64 "be9bcc194e
a402a4280ddde6f47e5a2d2c0dd8bfdd82d1982d81de0c1c9b0cea137c50beec4ceb70b528903600
6f1adf2cfd0e461bde43e561e68a71a2aa5819") }

16:36:06 00005-2110826240:STRTCloseSession {Sesn=2 }
16:36:06 00005-2110826240:FINICloseSession CKR_OK(3428288ms) {}
```

You can view the transaction by executing the following command on terminal 1:

```
# eth.pendingTransactions
```

```
> eth.pendingTransactions
[{
    blockHash: null,
    blockNumber: null,
    from: "0x474bca912e9f2fbbd3a0636402d876f87a0ec0cd",
    gas: 90000,
    gasPrice: 18000000000,
    hash: "0x0aeb8d18f3856da20b2f8a35e69c931cb33b2d2f5ceda390140a320e4ca2dba9",
    input: "0x",
    nonce: 0,
    r: "0xbe9bcc194ea402a4280ddde6f47e5a2d2c0dd8bfdd82d1982d81de0c1c9b0cea",
    s: "0x137c50beec4ceb70b5289036006f1adf2cfd0e461bde43e561e68a71a2aa5819",
    to: "0xd5feed8d1a1141a27f9545b32bf52e3f9316de53",
    transactionIndex: 0,
    v: "0x6fc",
    value: 500000000000000000
}]
```

6. Mine some more blocks in terminal 1.

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

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

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

7. In both consoles, observe the balance of the geth2 account.

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

   You should observe the balance transferred in the transaction.

   In above example, the balance in geth2 account should be: 500000000000000000

```
> eth.getBalance("0xd5feed8d1a1141a27f9545b32bf52e3f9316de53")
500000000000000000
```

8. Close the wallets.

   In terminal 1:

   ```
   # personal.closeWallet("hsm://geth1")
   ```

   In terminal 2:

   ```
   # personal.closeWallet("hsm://geth2")
   ```

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