Air Gap	= Cold Environment (Offline)
Hot Node	= Synchronized Node (Online)

Cardano-cli:~$ Study sheets

Part 2: stake pool, KES keys renewal and metadata

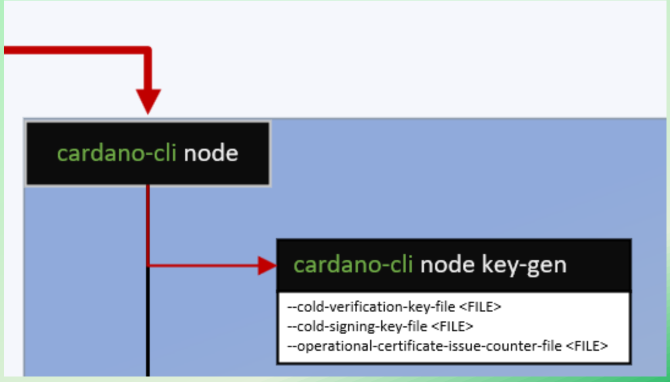
This tutorial is designed to be used with the Printable version of the Cardano-cli cheat sheet V8.0.0

The second part of this document is used to explain how to generate the three key pairs, the cold counter, the operational certificate for your pool. It will also explain how to register your pool and its metadata using the cardano-cli. We are going to simplify some commands that have already been explained in the first part of this document. We invite you to return to the previous exercises if the commands seem a little less familiar to you.

First exercise: Creation of the cold keys and the counter file

Air Gap

1 **First, locate the branch that you are going to use for your cold keys.**



cardano-cli node

cardano-cli node key-gen
--cold-verification-key-file <FILE>
--cold-signing-key-file <FILE>
--operational-certificate-issue-counter-file <FILE>

2 **There are 3 options in total each requiring the name for the new files.**

The cold keys are required to register a stake pool, to update a stake pool registration certificate parameters, to rotate a stake pool KES keys and to retire a stake pool. They are among the most important keys of your pool and should never leave your offline environment under any circumstances.

```
cardano-cli node key-gen
--cold-verification-key-file cold.vkey
--cold-signing-key-file cold.skey
--operational-certificate-issue-counter-file <FILE>
```

3 **There are 3 options in total each requiring the name for the new files.**

The counter file tracks the number of times an operational certificate has been generated for the relevant stake pool. We will explain all this in detail during the exercise about creating the operational certificate.

```
cardano-cli node key-gen
--cold-verification-key-file cold.vkey
--cold-signing-key-file cold.skey
--operational-certificate-issue-counter-file cold.counter
```

4 **This is what you will see on your terminal.**

```
user@computer:~$ cardano-cli node key-gen \
> --cold-verification-key-file cold.vkey \
> --cold-signing-key-file cold.skey \
> --operational-certificate-issue-counter-file cold.counter
```

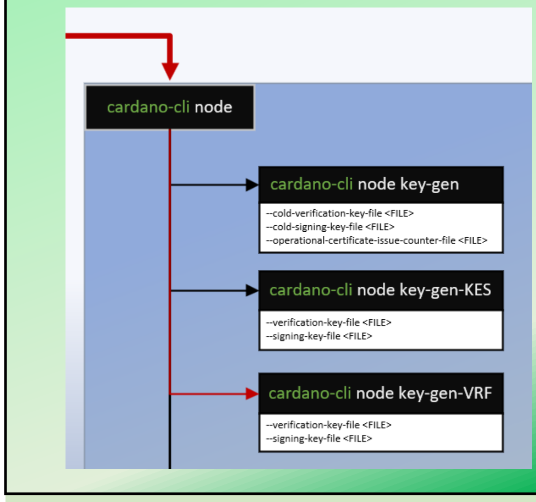
5 **When you open your cold.counter file, this is what you should see the first time. Pay particular attention to this. We'll talk about that later.**

```
{
    "type": "NodeOperationalCertificateIssueCounter",
    "description": "Next certificate issue number: 0",
    "cborHex": "82005820dcee462fec1d06f40ddbd6c120043a19b68beb384"
}
```

Second exercise: Creation of the VRF keys

Air Gap

1 **First, locate the branch that you are going to use for your VRF keys.**



cardano-cli node

cardano-cli node key-gen
--cold-verification-key-file <FILE>
--cold-signing-key-file <FILE>
--operational-certificate-issue-counter-file <FILE>

cardano-cli node key-gen-KES
--verification-key-file <FILE>
--signing-key-file <FILE>

cardano-cli node key-gen-VRF
--verification-key-file <FILE>
--signing-key-file <FILE>

2 **What exactly are VRF keys for?**

The Cardano network uses the Verifiable Random Function (VRF) to choose a random validator every epoch. In a vulgar way we can compare your VRF keys as your lottery ticket (ID) which will allow you to be chosen randomly as slot leader during an epoch in order to forge blocks. Your vrf.skey will also allow you to know the precise date and time when you will have the opportunity to forge them. (If you are chosen)

```
cardano-cli node key-gen-VRF
--verification-key-file <FILE>
--signing-key-file <FILE>
```

3 **So, nothing new, let's identify our 2 keys.**

```
cardano-cli node key-gen-VRF
--verification-key-file vrf.vkey
--signing-key-file vrf.skey
```

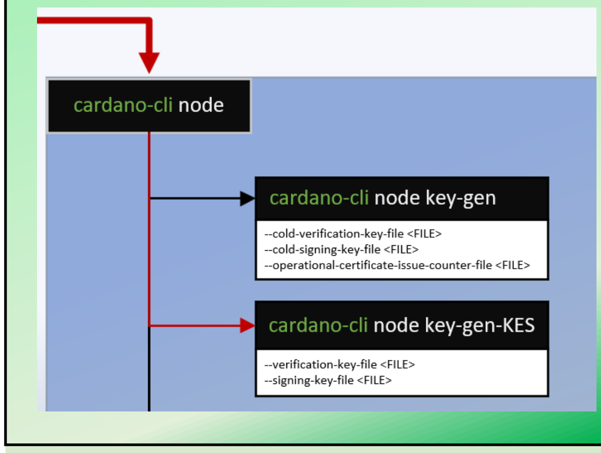
4 **This is what you will see on your terminal.**

```
user@computer:~$ cardano-cli node key-gen-VRF \
> --verification-key-file vrf.vkey \
> --signing-key-file vrf.skey \
```

Third exercise: Creation of the KES keys

Air Gap

1 **First, locate the branch that you are going to use for your KES keys.**



cardano-cli node

cardano-cli node key-gen
--cold-verification-key-file <FILE>
--cold-signing-key-file <FILE>
--operational-certificate-issue-counter-file <FILE>

cardano-cli node key-gen-KES
--verification-key-file <FILE>
--signing-key-file <FILE>

2 **What exactly are KES keys for?**

KES stands for Key Evolving Signature, which means that after a certain period, the key evolves to a new key and discards its old version, making it impossible for an attacker to rewrite history. You specify the validity of the KES key using the start time and key period parameters and this KES key needs to be updated every 90 days.(More explanation in the next exercise.)

```
cardano-cli node key-gen-KES
--verification-key-file <FILE>
--signing-key-file <FILE>
```

3 **Let's identify our 2 keys.**

```
cardano-cli node key-gen-KES
--verification-key-file kes.vkey
--signing-key-file kes.skey
```

4 **This is what you will see on your terminal.**

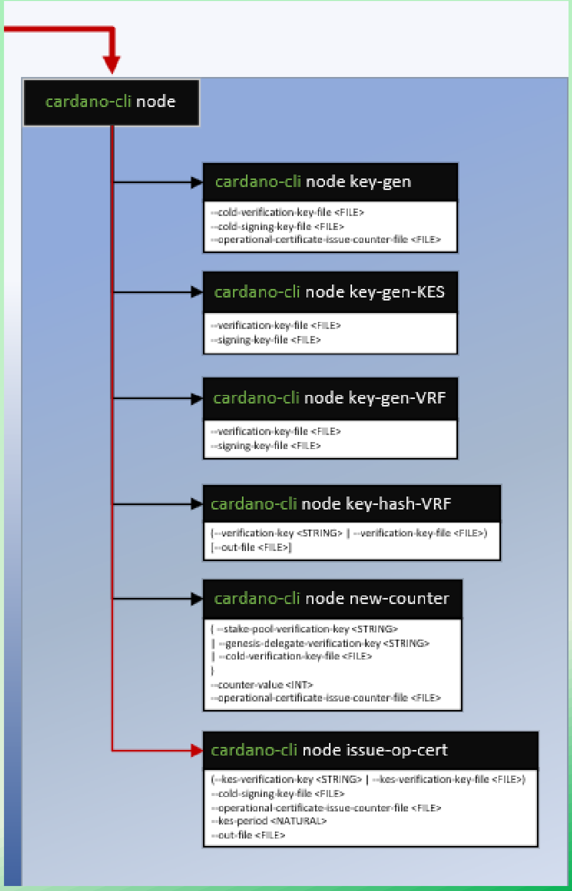
```
user@computer:~$ cardano-cli node key-gen-KES \
> --verification-key-file kes.vkey \
> --signing-key-file kes.skey \
```

Before moving on to the next exercise, you should know what an operational node certificate is. An operational node certificate represent the link between the operator's offline key and their operational key. A certificate's job is to check whether or not an operational key is valid, to prevent malicious interference. The certificate identifies the current operational key, and is signed by the offline key. (the cold.skey)

Fourth exercise: Creation of the operational node certificate

Air Gap

1 **First, locate the branch that you are going to use for your operational certificate.**


cardano-cli node
cardano-cli node key-gen
--cold-verification-key-file <FILE>
--cold-signing-key-file <FILE>
--operational-certificate-issue-counter-file <FILE>
cardano-cli node key-gen-KES
--verification-key-file <FILE>
--signing-key-file <FILE>
cardano-cli node key-gen-VRF
--verification-key-file <FILE>
--signing-key-file <FILE>
cardano-cli node key-hash-VRF
(--verification-key <STRING> | --verification-key-file <FILE>)
[--out-file <FILE>]
cardano-cli node new-counter
(--stake-pool-verification-key <STRING>
| --genesis-delegate-verification-key <STRING>
| --cold-verification-key-file <FILE>
)
--counter-value <INT>
--operational-certificate-issue-counter-file <FILE>
cardano-cli node issue-op-cert
(--kes-verification-key <STRING> | --kes-verification-key-file <FILE>)
--cold-signing-key-file <FILE>
--operational-certificate-issue-counter-file <FILE>
--kes-period <NATURAL>
--out-file <FILE>

2 **Again,an operational certificate job is to check whether or not an operational key is valid to prevent malicious interference. Which is why it needs 4 things.**

cardano-cli node issue-op-cert

```
(--kes-verification-key <STRING> | --kes-verification-key-file <FILE>)
--cold-signing-key-file <FILE>
--operational-certificate-issue-counter-file <FILE>
--kes-period <NATURAL>
--out-file <FILE>
```

- **The KES verification key** of the operational key its going to valid.
- **The cold key signature** to make the link between the operator's offline key and the operational key.
- **The counter file** to track the certificate issue number. (more on that later)
- **The KES current period** to validate the KES key, its evolution and its expiration.

3 **For now you can specify the PATH to your kes.vkey, your cold.skey and your cold.counter.**

For the '--kes-period' option, you must determine the KES period in which we are currently in order to be able to prove the validity of your KES key, follow its evolution and at the same time, know its expiration date. To do this, you will need to know the current slot number by performing the command specified in part 1, exercise 6 of this document. (cardano-cli query tip)

cardano-cli node issue-op-cert

(~~--kes-verification-key <STRING>~~ | --kes-verification-key-file **kes.vkey**)
--cold-signing-key-file **cold.skey**
--operational-certificate-issue-counter-file **cold.counter**
--kes-period <NATURAL>
--out-file <FILE>

4 **Here is an example of the result of the command (cardano-cli query tip)**

```
user@computer:~$ cardano-cli query tip \
> --mainnet
```

```
{
  "block": 8749305,
  "epoch": 411,
  "era": "Babbage",
  "hash": "505e4af96abc19e1d8e0d54cb508e564...",
  "slot": 94027764,
  "syncProgress": "100.00"
}
```

5 **You also need to know how many slots there are per KES period. (1 slot = 1 second)**

This information is written in the last few lines of your node shelley-genesis.json file

```
"networkId": "Mainnet",
"initialFunds": {},
"maxLovelaceSupply": 45000000000000000,
"networkMagic": 764824073,
"epochLength": 432000,
"systemStart": "2017-09-23T21:44:51Z",
"slotsPerKESPeriod": 129600,
"slotLength": 1,
"maxKESEvolutions": 62,
```

6 **You now know the current slot number and the number of "slotsPerKESperiod"**

You can calculate what KES period you are currently in by dividing the current slot number (94027764) by the "slotsPerKESPeriod" (12960)

```
user@computer:~$ expr 94027764 / 129600
725
```

KES actual period

7 **Add it to your cardano-cli command as well as the name of your operational node certificate.**

cardano-cli node issue-op-cert

(~~--kes-verification-key <STRING>~~ | --kes-verification-key-file **kes.vkey**)
--cold-signing-key-file **cold.skey**
--operational-certificate-issue-counter-file **cold.counter**
--kes-period **725**
--out-file **node.cert**

8 **This is what you will see on your terminal.**

```
user@computer:~$ cardano-cli node issue-op-cert \
> --kes-verification-key-file kes.vkey \
> --cold-signing-key-file cold.skey \
> --operational-certificate-issue-counter-file cold.counter \
> --kes-period 725 \
> --out-file node.cert
```

9 **Now that you have created your node.cert file (which have a counter of 0), check the content of your cold.counter file and notice what has just changed.**

```
{
  "type": "NodeOperationalCertificateIssueCounter",
  "description": "Next certificate issue number: 1",
  "cborHex": "134553546cee462fec1df5440ddbd6c11453a19b68bac5678"
}
```

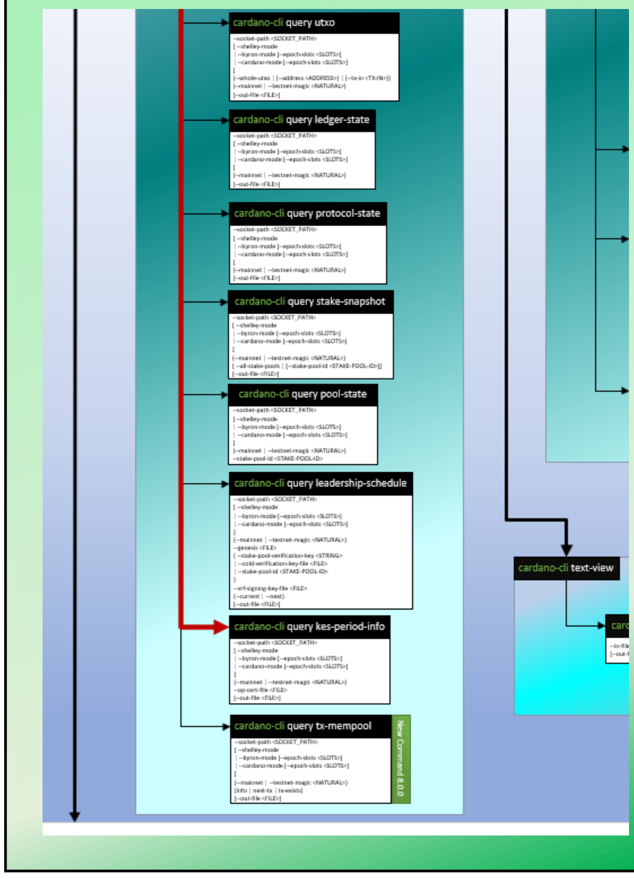
This number has changed from 0 to 1

You can now transfer your kes.skey, vrf.skey and your node.cert to the node that will serve as your block producer. What will follow will be extremely important to understand. If you are unable to properly renew your KES key when it expires, you could lose your blocks despite the fact that they have been assigned to you and they will be considered invalid until you rectify the situation. To do this, you will need to understand how it works and the role of the counter file when renewing your node.cert and your KES keys.

Fifth exercise: Understanding of KES key renewal

Hot Node | Air Gap

1 **First, we'll need to run the ''cardano-cli query kes-period-info'' command on your hot node.**


cardano-cli query utxo
cardano-cli query ledger-state
cardano-cli query protocol-state
cardano-cli query stake-snapshot
cardano-cli query pool-state
cardano-cli query leadership-schedule
cardano-cli query kes-period-info
cardano-cli query tx-mempool
New Command 8.0.0
cardano-cli text-view

2 **Nothing too complicated, you will only use the 2 mandatory options among these.**

cardano-cli query kes-period-info

~~--socket-path <SOCKET_PATH>~~
~~[--shelley-mode~~
~~| --byron-mode [--epoch-slots <SLOTS>]~~
~~| --cardano-mode [--epoch-slots <SLOTS>]~~
]
(--mainnet | ~~--testnet-magic <NATURAL>)~~
--op-cert-file **node.cert**
~~[--out-file <FILE>]~~

3 **This command will allow us to obtain information on the current KES period and on our operational certificate.**

```
user@computer:~$ cardano-cli query kes-period-info \
> --mainnet \
> --op-cert-file node.cert
```

4 **This is the ideal situation because "qKesNodeStateOperationalCertificateNumber" and "qKesOnDiskOperationalCertificateNumber" have exactly the same number.**

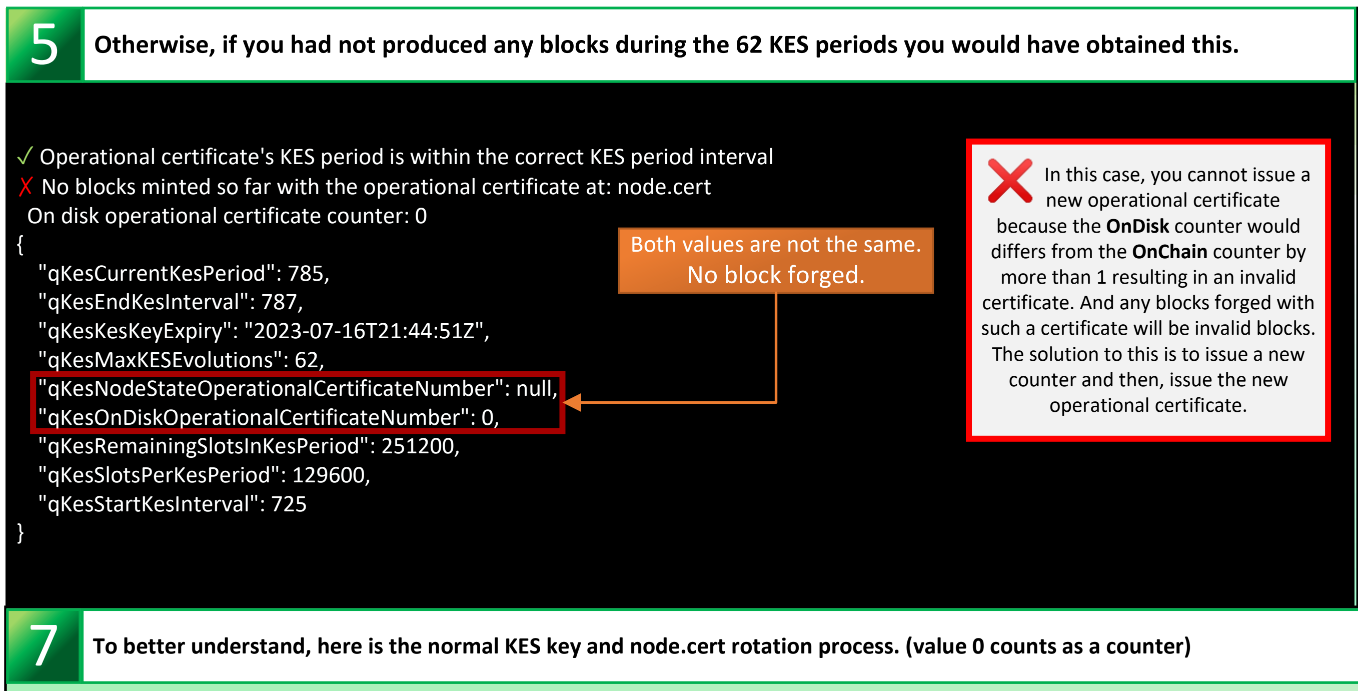
```
✓ Operational certificate's KES period is within the correct KES period interval
✓ The operational certificate counter agrees with the node protocol state counter
{
  "qKesCurrentKesPeriod": 785,
  "qKesEndKesInterval": 787,
  "qKesKesKeyExpiry": "2023-07-16T21:44:51Z",
  "qKesMaxKESEvolutions": 62,
  "qKesNodeStateOperationalCertificateNumber": 0,
  "qKesOnDiskOperationalCertificateNumber": 0,
  "qKesRemainingSlotsInKesPeriod": 251200,
  "qKesSlotsPerKesPeriod": 129600,
  "qKesStartKesInterval": 725
}
```

last counter registered OnChain **(KesNodeState)**

Counter value on your disk **(KesOnDisk)**

✓ When your pool has forged at least one block with the current operational certificate the values will match. So, in this particular case, the pool produced one or more blocks during the 62 KES periods of its operational certificate. You just have to renew your KES keys and make new node.cert in your "Air Gap" environment. Then transfer them to your block producer and you're done. (repeat exercise 3 and 4 of this part)

5 Otherwise, if you had not produced any blocks during the 62 KES periods you would have obtained this.



```
✓ Operational certificate's KES period is within the correct KES period interval
✗ No blocks minted so far with the operational certificate at: node.cert
 On disk operational certificate counter: 0
{
    "qKesCurrentKesPeriod": 785,
    "qKesEndKesInterval": 787,
    "qKesKesKeyExpiry": "2023-07-16T21:44:51Z",
    "qKesMaxKESEvolutions": 62,
    "qKesNodeStateOperationalCertificateNumber": null,
    "qKesOnDiskOperationalCertificateNumber": 0,
    "qKesRemainingSlotsInKesPeriod": 251200,
    "qKesSlotsPerKesPeriod": 129600,
    "qKesStartKesInterval": 725
}
```

Both values are not the same.
No block forged.

In this case, you cannot issue a new operational certificate because the **OnDisk** counter would differs from the **OnChain** counter by more than 1 resulting in an invalid certificate. And any blocks forged with such a certificate will be invalid blocks. The solution to this is to issue a new counter and then, issue the new operational certificate.

6 So that you can fully understand how these counters work, we will give you a chronological example.

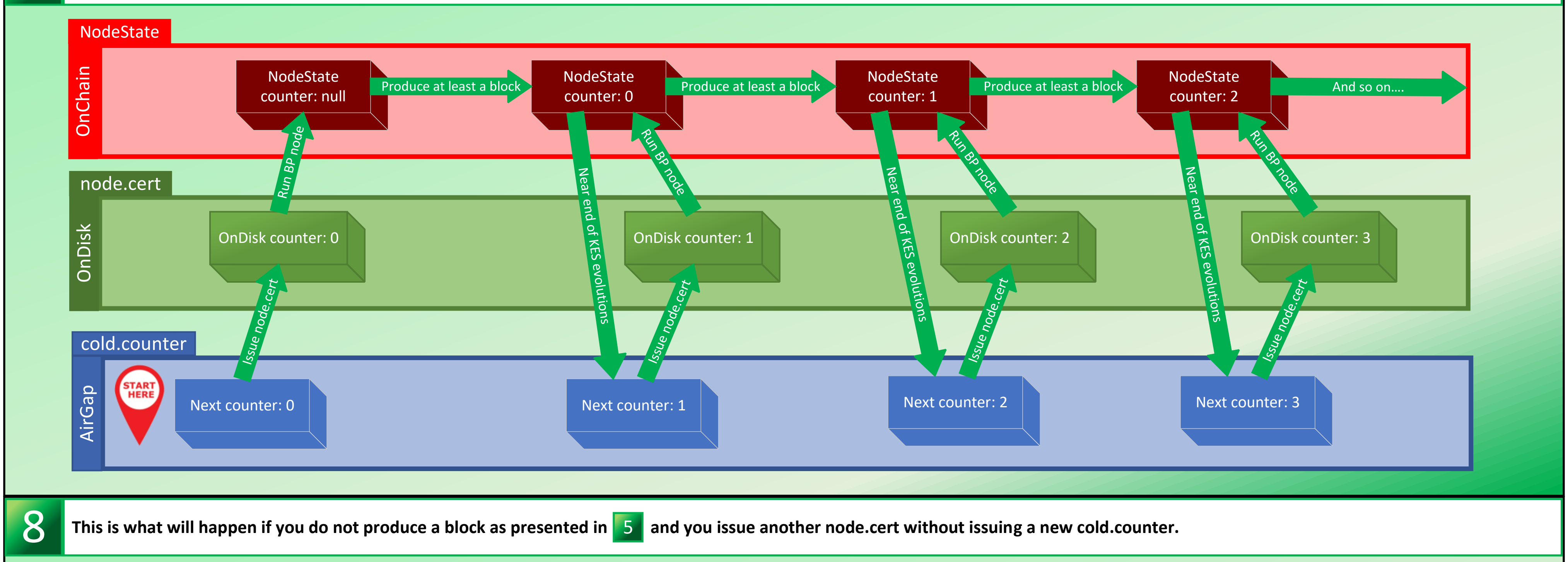
If you observe at the very beginning of the part 2, even before having created your first operational certificate on your''Air Gap''computer . Your cold.counter was indicating that the next certificate issue number would be 0.

```
{
    "type": "NodeOperationalCertificateIssueCounter",
    "description": "Next certificate issue number: 0",
    "cborHex": "82005820dcee462fec1d06f40dd120048beb384..."
}
```

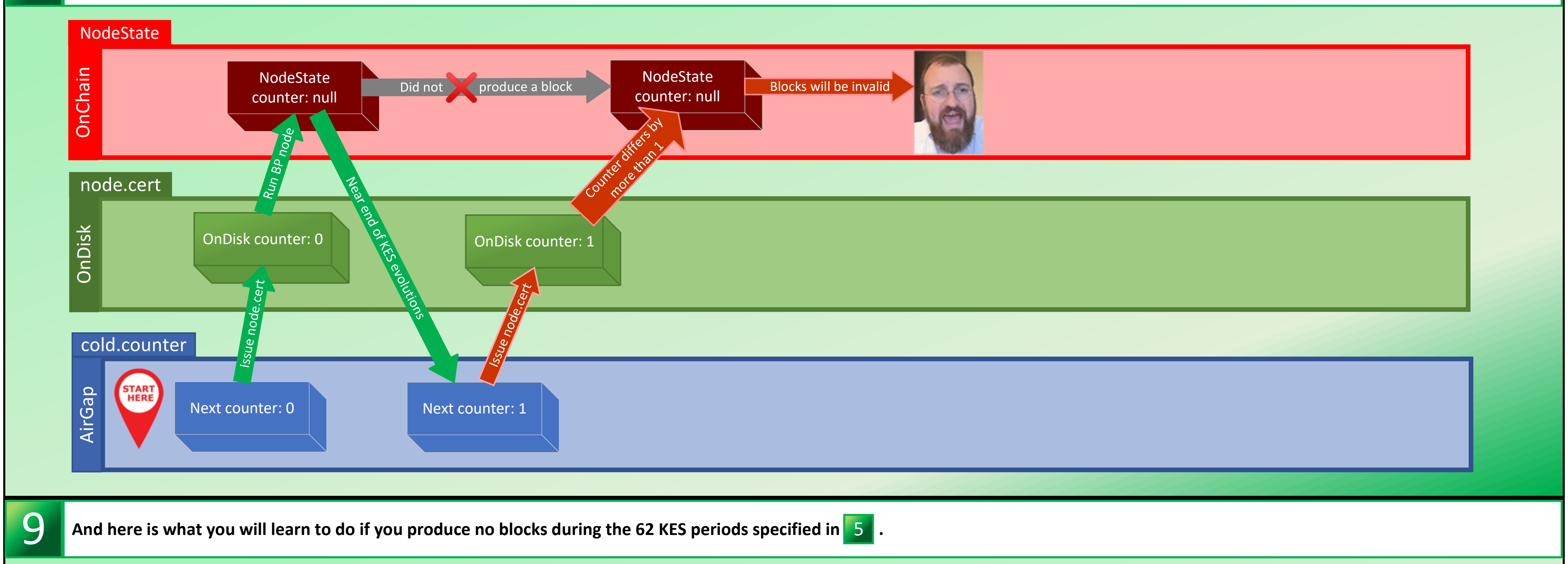
Then when you create your first node.cert (which will have a value of 0), your cold.counter will change accordingly for your next renewal and so on.

```
{
    "type": "NodeOperationalCertificateIssueCounter",
    "description": "Next certificate issue number: 1",
    "cborHex": "134553546cee462fe1df5440ddb19b68bac5678..."
}
```

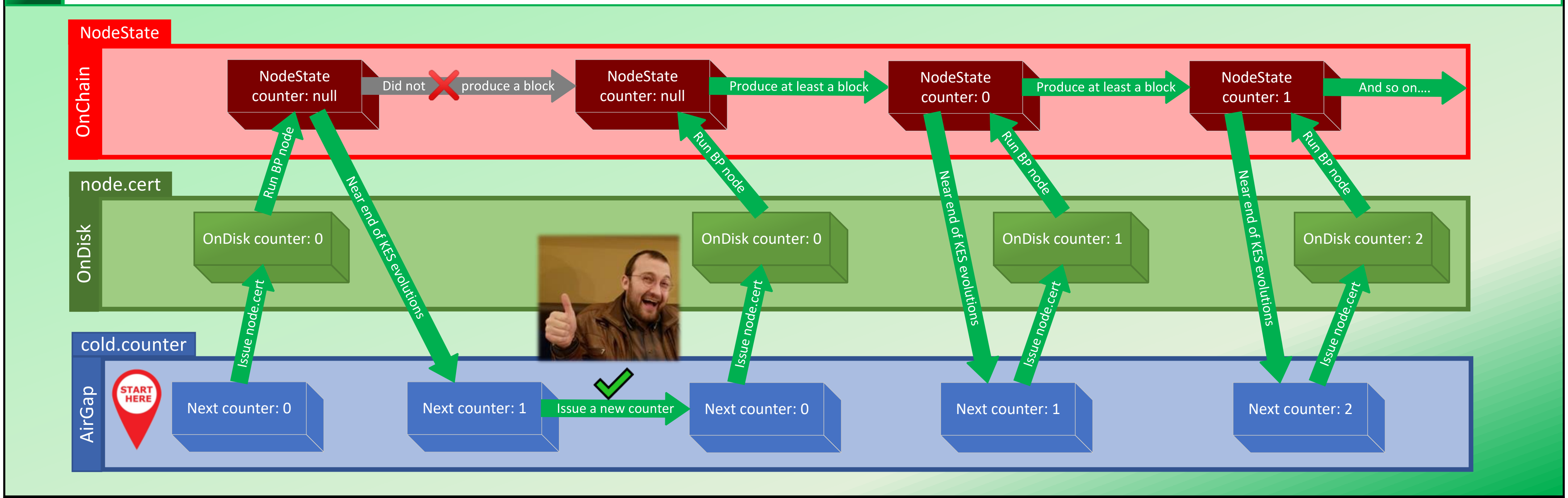
7 To better understand, here is the normal KES key and node.cert rotation process. (value 0 counts as a counter)



8 This is what will happen if you do not produce a block as presented in 5 and you issue another node.cert without issuing a new cold.counter.



9 And here is what you will learn to do if you produce no blocks during the 62 KES periods specified in 5 .



10 This is how to fix problem 5 .

You cannot use this cold.counter file to issue a new node.cert because its counter value will be 1 if you do. And this value will be 2 more than the NodeState counter value.

```
{
    "type": "NodeOperationalCertificateIssueCounter",
    "description": "Next certificate issue number: 1",
    "cborHex": "134553546cee462fe1df5440ddb19b68bac5678..."
}
```

You need to issue a new cold.counter and increment it by exactly 1 more than ''qKesNodeStateOperationalCertificateNumber'' value.

11 To do this you will use the command "cardano-cli node new-counter" on your ''Air Gap'' environment.

```
cardano-cli node new-counter

( --stake-pool-verification-key <STRING>
| --genesis-delegate-verification-key <STRING>
| --cold-verification-key-file <FILE>
)
--counter-value <INT>
--operational-certificate-issue-counter-file <FILE>
```

12 Use the mandatory options of your choice among the first 3. Also use your actual cold.counter.

cardano-cli node new-counter

~~(--stake-pool-verification-key <STRING>~~
~~| --genesis-delegate-verification-key <STRING>~~
| --cold-verification-key-file **cold.vkey**
)
--counter-value <INT>
--operational-certificate-issue-counter-file **cold.counter**

13 **Next, Take the OnChain counter and increment it by exactly 1, In our case, it will be 0. (null+1=0)**

```
cardano-cli node new-counter
~~( --stake-pool-verification-key <STRING>~~
~~| --genesis-delegate-verification-key <STRING>~~
| --cold-verification-key-file cold.vkey
)
--counter-value 0
--operational-certificate-issue-counter-file cold.counter
```

14 **This is what you will see on your terminal.**

```
user@computer:~$ cardano-cli node new-counter \
> --cold-verification-key-file cold.vkey \
> --counter-value 0 \
> --operational-certificate-issue-counter-file cold.counter
```

15 **Do not panic!**

When you check your cold.counter file after changing its counter, you should see this. Don't worry, the counter has been successfully issued.

```
{
   "type": "NodeOperationalCertificateIssueCounter",
   "description": "",
   "cborHex": "82005820dcee462fec1d06f40dd120048beb384..."
}
```

16 **As soon as it is used, the cold.counter description will return to normal.**

So by repeating the procedure of the third and fourth excercise to renew your KES key and your node.cert, the value 0 of your cold.counter will be used and will increase by 1 for your next renewal.

```
{
   "type": "NodeOperationalCertificateIssueCounter",
   "description": "Next certificate issue number: 1",
   "cborHex": "134553546cee462fe1df5440ddb19b68bac5678..."
}
```

You can now transfer your renewed kes.skey and node.cert to your block producer node and check their validity with the "cardano-cli query kes-period-info" command from the beginning of this exercise.

17 **Here is an example of what you could get.**

```
✓ Operational certificate's KES period is within the correct KES period interval
✗ No blocks minted so far with the operational certificate at: node.cert
 On disk operational certificate counter: 0
{
  "qKesCurrentKesPeriod": 785,
  "qKesEndKesInterval": 847,
  "qKesKesKeyExpiry": "2023-08-12T20:54:51Z",
  "qKesMaxKESEvolutions": 62,
  "qKesNodeStateOperationalCertificateNumber": null,
  "qKesOnDiskOperationalCertificateNumber": 0,
  "qKesRemainingSlotsInKesPeriod": 8035005,
  "qKesSlotsPerKesPeriod": 129600,
  "qKesStartKesInterval": 785
}
```

✓ Make sure that the value of **"qKesStartKesInterval"** is equal to that of **"qKesCurrentKesPeriod"**

✓ Also make sure that the value of **"qKesOnDiskOperationalCertificateNumber"** is greater than the value of **"qKesNodeStateOperationalCertificateNumber"** by exactly one. And that's it.

You can now transfer your kes.skey, vrf.skey(if needed) and your node.cert to the node that will serve as your block producer and start it. You still have to generate your pool metadata and submit your stake pool certificate so that your pool becomes visible to all cardano wallets available. This way, people from the community will finally be able to stake their ada to your pool.

Sixth exercise: Pool Metadata and metadata hash.

Hot Node | Air Gap

1 **First, you must create your pool's metadata and make it available in .json format on a domain(URL) that you maintain. Here is the pattern to use:**

```
{
"name": "TestPool",
"description": "The pool that tests all the pools",
"ticker": "TEST",
"homepage": "https://teststakepool.com"
}
```

Your pool's metadata contains information about your pool such as: its name, its ticker, a description of the pool and a website link which will appear in each cardano wallets.
Ensure that the Stake pool metadata consists of at most 512 bytes, with the URL being less than 65 characters long.

2 **This is the command to perform to know the hash of your pool's metadata that you will have to use for your stake pool certificate in the next exercise.**

```
cardano-cli stake-pool metadata-hash
--pool-metadata-file <FILE>
~~[--out-file <FILE>]~~
```

On your server

```
user@computer:~$ cardano-cli stake-pool metadata-hash \
> --pool-metadata-file testpool.json
1ee45c2686de8429c831300d2de4cc2afef579fbe4c6e7355f0c76f8b8829d95
```

On your domain (URL)

```
user@computer:~$ cardano-cli stake-pool metadata-hash \
> --pool-metadata-file  <(curl -s -L your_URL_link_to_Metadata )
1ee45c2686de8429c831300d2de4cc2afef579fbe4c6e7355f0c76f8b8829d95
```

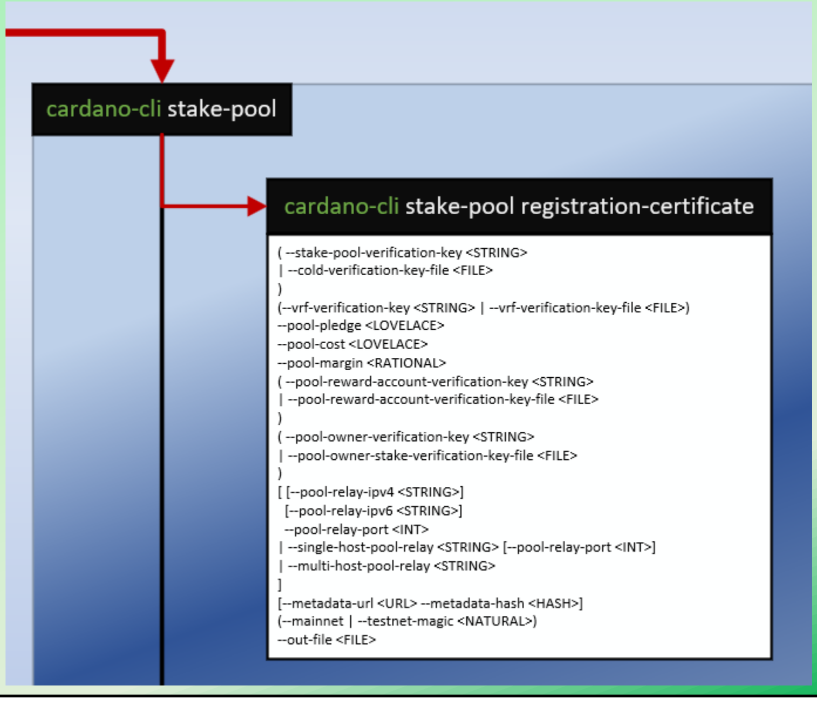
Once your metadata is submitted onchain with the stake pool certificate, you will be able to check if your metadata is valid and if your URL resolves to the metadata hash submitted with SMASH. Cardano Stakepool Metadata Aggregation Server (SMASH) is a server that aggregates common metadata about registered stakepools on the Cardano blockchain, such as the name of the stakepool, its "ticker" name, and homepage. SMASH aims to ensure that registered stake pools are valid, avoid duplicated ticker names or trademarks. (More about that after the next exercise.

Seventh exercise: Creating the stake pool certificate

Air Gap

1 **First, locate the branch that you are going to use for your stake pool registration certificate.**

This one may seem a little confusing but we won't let you down. We will get through this together and in a simple and understandable way.



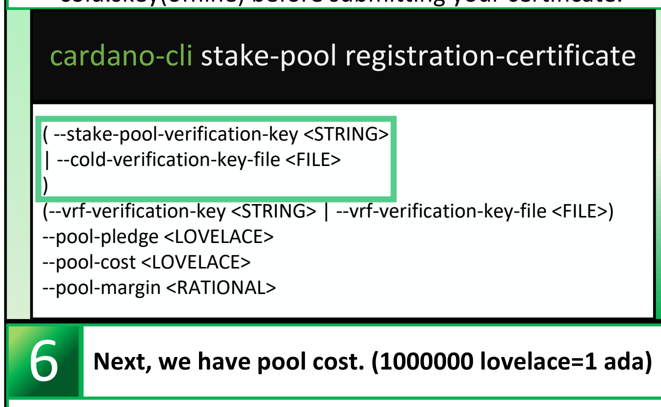
```
cardano-cli stake-pool

cardano-cli stake-pool registration-certificate
( --stake-pool-verification-key <STRING>
| --cold-verification-key-file <FILE>
)
(--vrf-verification-key <STRING> | --vrf-verification-key-file <FILE>)
--pool-pledge <LOVELACE>
--pool-cost <LOVELACE>
--pool-margin <RATIONAL>
( --pool-reward-account-verification-key <STRING>
| --pool-reward-account-verification-key-file <FILE>
)
( --pool-owner-verification-key <STRING>
| --pool-owner-stake-verification-key-file <FILE>
)
[ [--pool-relay-ipv4 <STRING>]
  [--pool-relay-ipv6 <STRING>]
  --pool-relay-port <INT>
| --single-host-pool-relay <STRING> [--pool-relay-port <INT>]
| --multi-host-pool-relay <STRING>
]
[--metadata-url <URL> --metadata-hash <HASH>]
(--mainnet | --testnet-magic <NATURAL>)
--out-file <FILE>
```

2 **There are 21 options in total with priority columns for relay options. We will explain them all one by one.**

```
cardano-cli stake-pool registration-certificate
( --stake-pool-verification-key <STRING>
| --cold-verification-key-file <FILE>
)
(--vrf-verification-key <STRING> | --vrf-verification-key-file <FILE>)
--pool-pledge <LOVELACE>
--pool-cost <LOVELACE>
--pool-margin <RATIONAL>
( --pool-reward-account-verification-key <STRING>
| --pool-reward-account-verification-key-file <FILE>
)
( --pool-owner-verification-key <STRING>
| --pool-owner-stake-verification-key-file <FILE>
)
[ [--pool-relay-ipv4 <STRING>]
  [--pool-relay-ipv6 <STRING>]
  --pool-relay-port <INT>
| --single-host-pool-relay <STRING> [--pool-relay-port <INT>]
| --multi-host-pool-relay <STRING>
]
[--metadata-url <URL> --metadata-hash <HASH>]
(--mainnet | --testnet-magic <NATURAL>)
--out-file <FILE>
```

3 First set of options:

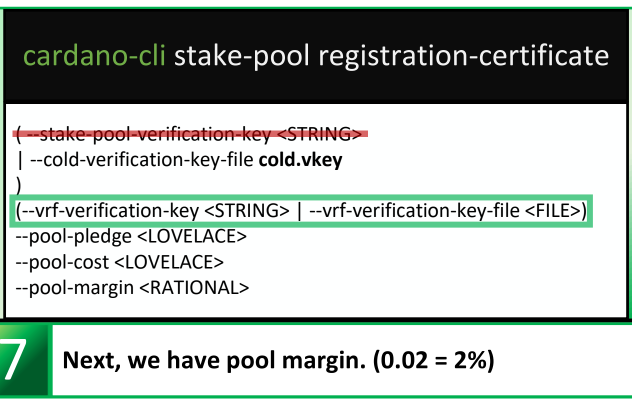
Your pool ID will be taken from your cold verification key, which will of course need to be signed by your cold.skey(offline) before submitting your certificate.



```
cardano-cli stake-pool registration-certificate

( --stake-pool-verification-key <STRING>
| --cold-verification-key-file <FILE>
)
(--vrf-verification-key <STRING> | --vrf-verification-key-file <FILE>)
--pool-pledge <LOVELACE>
--pool-cost <LOVELACE>
--pool-margin <RATIONAL>
```

4 Next is vrf-verification-key-file

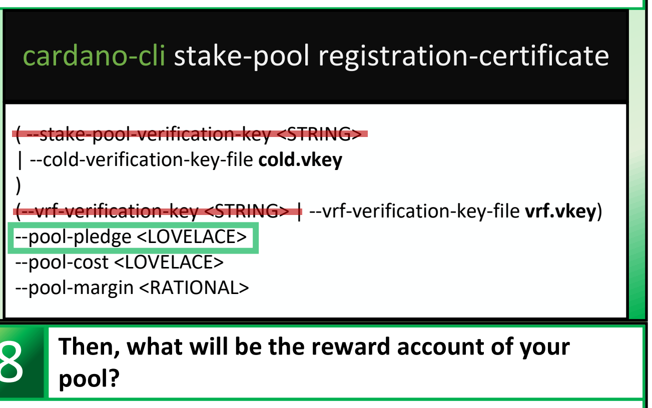
linked to your certificate, it will be your lottery ticket for your leader slots assignements. (if you get any)



```
cardano-cli stake-pool registration-certificate

~~( --stake-pool-verification-key <STRING>~~
| --cold-verification-key-file cold.vkey
)
(--vrf-verification-key <STRING> | --vrf-verification-key-file <FILE>)
--pool-pledge <LOVELACE>
--pool-cost <LOVELACE>
--pool-margin <RATIONAL>
```

5 Next, we have pool pledge. (in lovelace)

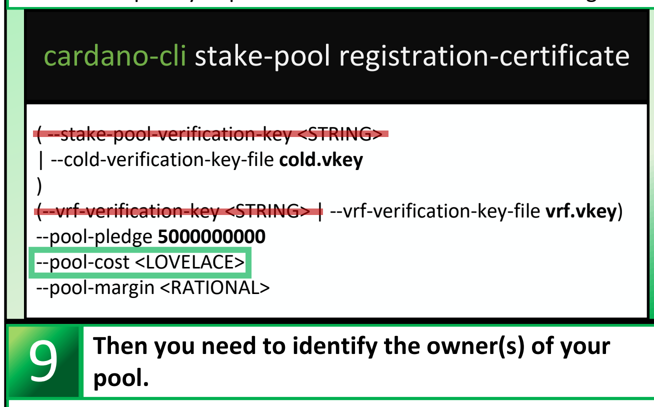
The pool pledge refers to the amount of ADA that a stake pool owner commits to delegate to their pool.



```
cardano-cli stake-pool registration-certificate

~~( --stake-pool-verification-key <STRING>~~
| --cold-verification-key-file cold.vkey
)
~~(--vrf-verification-key <STRING> |~~ --vrf-verification-key-file vrf.vkey)
--pool-pledge <LOVELACE>
--pool-cost <LOVELACE>
--pool-margin <RATIONAL>
```

6 Next, we have pool cost. (1000000 lovelace=1 ada)

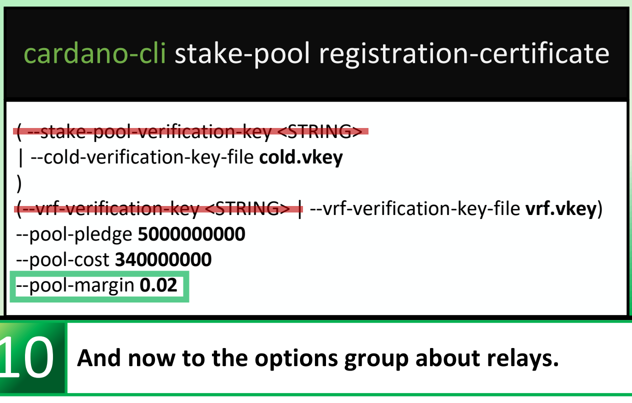
The pool cost across the Cardano network is 340 ADA minimum. It cannot be any lower. This is the minimum amount you will have each epoch you produce blocks. You can also set it higher.



```
cardano-cli stake-pool registration-certificate

~~( --stake-pool-verification-key <STRING>~~
| --cold-verification-key-file cold.vkey
)
~~(--vrf-verification-key <STRING> |~~ --vrf-verification-key-file vrf.vkey)
--pool-pledge 5000000000
--pool-cost <LOVELACE>
--pool-margin <RATIONAL>
```

7 Next, we have pool margin. (0.02 = 2%)

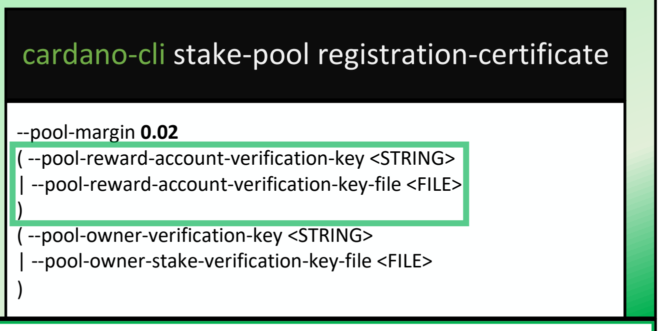
Pool margin in Cardano refers to the variable margin fee set by the pool operator, which is usually between 0%-10%.



```
cardano-cli stake-pool registration-certificate

~~( --stake-pool-verification-key <STRING>~~
| --cold-verification-key-file cold.vkey
)
~~(--vrf-verification-key <STRING> |~~ --vrf-verification-key-file vrf.vkey)
--pool-pledge 5000000000
--pool-cost 340000000
--pool-margin 0.02
```

8 Then, what will be the reward account of your pool?

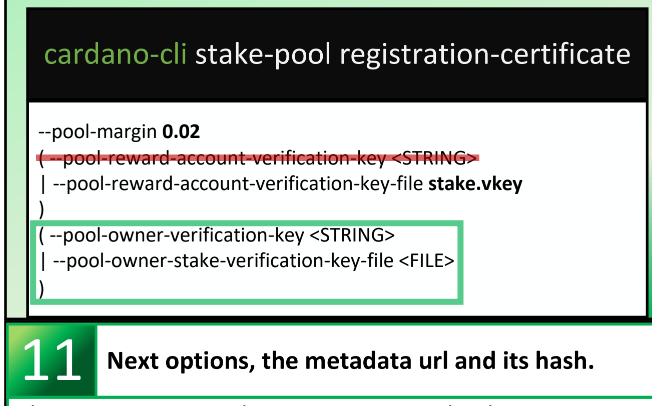
You can use your stake.vkey or the staking address of your choice for your pool rewards.



```
cardano-cli stake-pool registration-certificate

--pool-margin 0.02
( --pool-reward-account-verification-key <STRING>
| --pool-reward-account-verification-key-file <FILE>
)
( --pool-owner-verification-key <STRING>
| --pool-owner-stake-verification-key-file <FILE>
)
```

9 Then you need to identify the owner(s) of your pool.

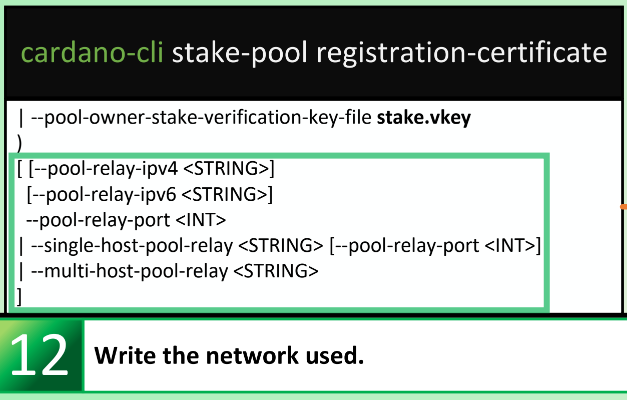
You can add your stake.vkey. You can also add multiple owners by repeating the option.



```
cardano-cli stake-pool registration-certificate

--pool-margin 0.02
~~( --pool-reward-account-verification-key <STRING>~~
| --pool-reward-account-verification-key-file stake.vkey
)
( --pool-owner-verification-key <STRING>
| --pool-owner-stake-verification-key-file <FILE>
)
```

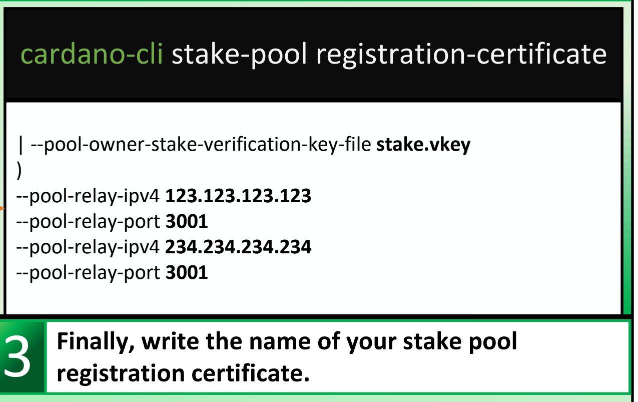
10 And now to the options group about relays.

You can have one or more public relays. Simply repeat the chosen option for each relay. You can declare their address in ipv4, ipv6, single host (DNS) or multi-host (DNS). For each address, you must mention the port number of your node. In our example below, the operator declared 2 public relays. ****WARNING! Do not declare the ip address of your block producer. Only your public relays.****



```
cardano-cli stake-pool registration-certificate

| --pool-owner-stake-verification-key-file stake.vkey
)
[ [--pool-relay-ipv4 <STRING>]
  [--pool-relay-ipv6 <STRING>]
  --pool-relay-port <INT>
| --single-host-pool-relay <STRING> [--pool-relay-port <INT>]
| --multi-host-pool-relay <STRING>
]
```

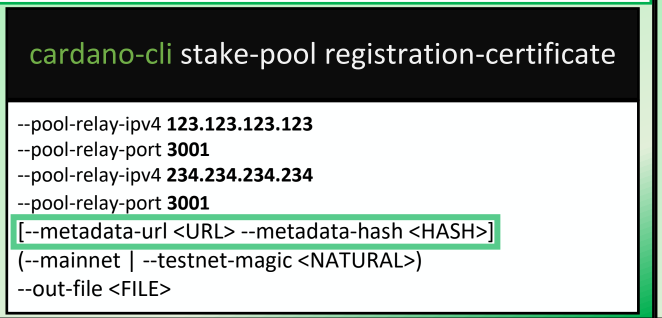
→



```
cardano-cli stake-pool registration-certificate

| --pool-owner-stake-verification-key-file stake.vkey
)
--pool-relay-ipv4 123.123.123.123
--pool-relay-port 3001
--pool-relay-ipv4 234.234.234.234
--pool-relay-port 3001
```

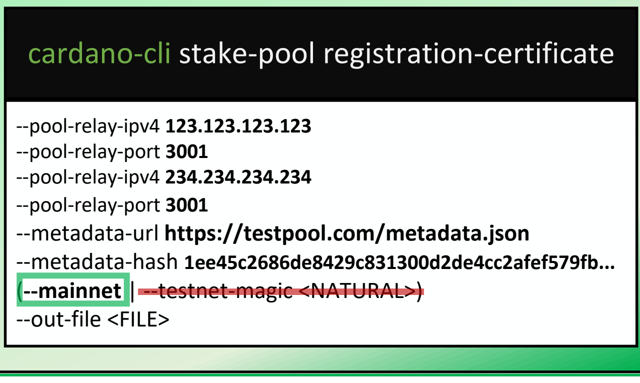
11 Next options, the metadata url and its hash.

This option is not mandatory since you can decide to operate a private stake pool with a closed circle of investors. Otherwise, use your metadata URL and its hash as practiced in exercise 6.



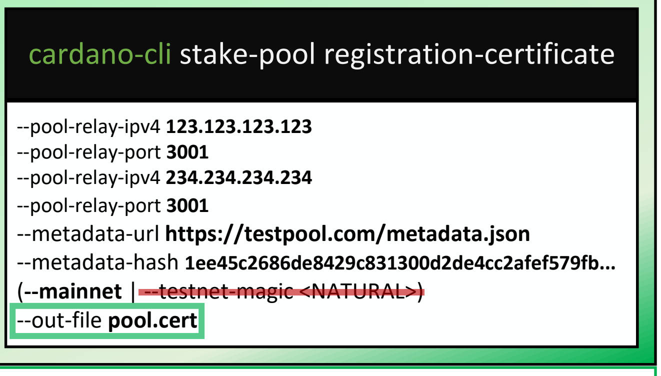
```
cardano-cli stake-pool registration-certificate

--pool-relay-ipv4 123.123.123.123
--pool-relay-port 3001
--pool-relay-ipv4 234.234.234.234
--pool-relay-port 3001
[--metadata-url <URL> --metadata-hash <HASH>]
(--mainnet | --testnet-magic <NATURAL>)
--out-file <FILE>
```

12 Write the network used.



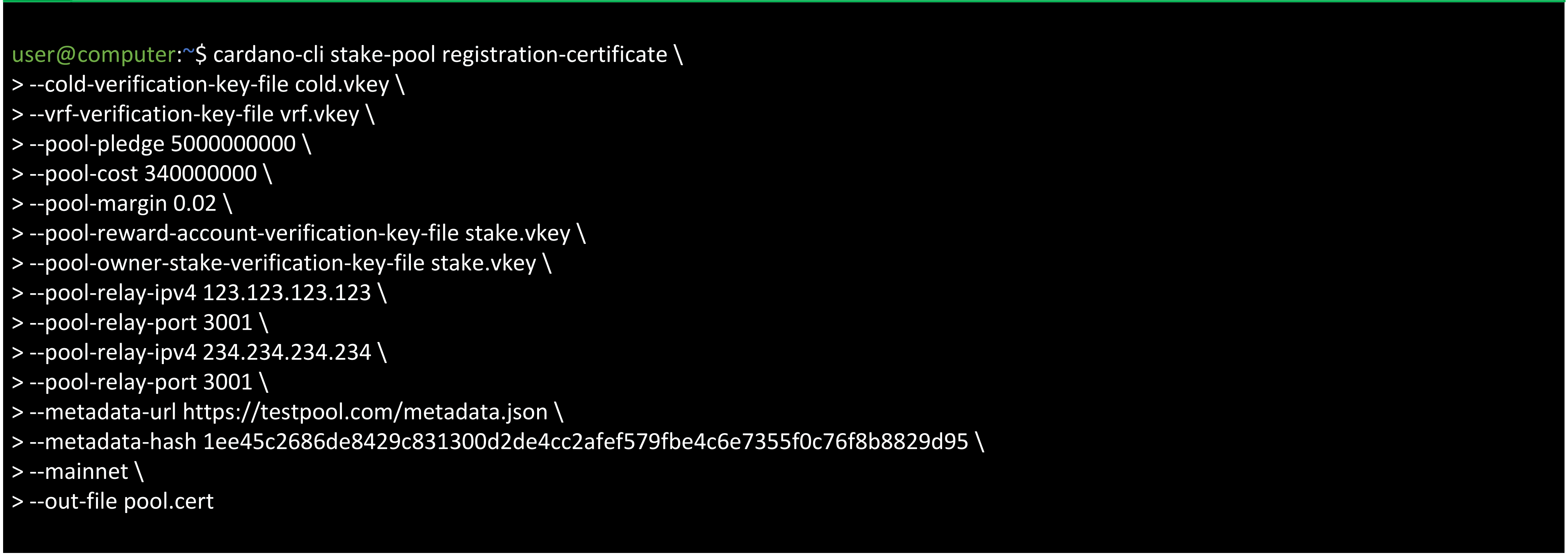
```
cardano-cli stake-pool registration-certificate

--pool-relay-ipv4 123.123.123.123
--pool-relay-port 3001
--pool-relay-ipv4 234.234.234.234
--pool-relay-port 3001
--metadata-url https://testpool.com/metadata.json
--metadata-hash 1ee45c2686de8429c831300d2de4cc2afef579fb...
(--mainnet | ~~--testnet-magic <NATURAL>)~~
--out-file <FILE>
```

13 Finally, write the name of your stake pool registration certificate.



```
cardano-cli stake-pool registration-certificate

--pool-relay-ipv4 123.123.123.123
--pool-relay-port 3001
--pool-relay-ipv4 234.234.234.234
--pool-relay-port 3001
--metadata-url https://testpool.com/metadata.json
--metadata-hash 1ee45c2686de8429c831300d2de4cc2afef579fb...
(--mainnet | ~~--testnet-magic <NATURAL>)~~
--out-file pool.cert
```

14 This is what you will see on your terminal.



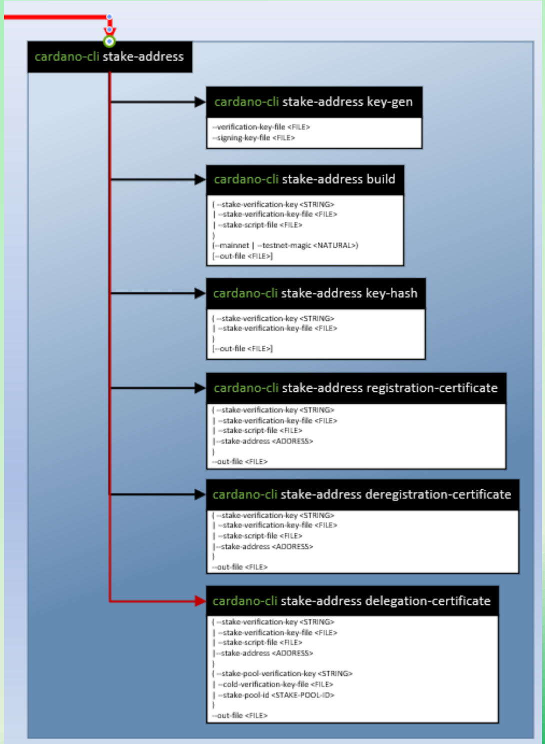
```
user@computer:~$ cardano-cli stake-pool registration-certificate \
> --cold-verification-key-file cold.vkey \
> --vrf-verification-key-file vrf.vkey \
> --pool-pledge 5000000000 \
> --pool-cost 340000000 \
> --pool-margin 0.02 \
> --pool-reward-account-verification-key-file stake.vkey \
> --pool-owner-stake-verification-key-file stake.vkey \
> --pool-relay-ipv4 123.123.123.123 \
> --pool-relay-port 3001 \
> --pool-relay-ipv4 234.234.234.234 \
> --pool-relay-port 3001 \
> --metadata-url https://testpool.com/metadata.json \
> --metadata-hash 1ee45c2686de8429c831300d2de4cc2afef579fbe4c6e7355f0c76f8b8829d95 \
> --mainnet \
> --out-file pool.cert
```

Now that your stake pool registration certificate is ready, you will need to redo a delegation certificate in order to stake to your own pool with your main wallet. But don't worry, because thanks to Cardano's UTXO transaction model, you can submit them at the same time in the same transaction. To do this, you will have to use the method in "part 1, exercise 13" of this document but in a slightly different way this time. Since your pool is technically still not registered onchain, it will be difficult for you to find your pool ID on cexplorer.io to stake the ada of your pool owner wallet. So here's how to do it:

Eighth exercise: Creation of a delegation certificate for stake pool owners

Air Gap

1 First, locate the branch that you are going to use for your certificate.



cardano-cli stake-address

cardano-cli stake-address key-gen

```
--verification-key-file <FILE>
--signing-key-file <FILE>
```

cardano-cli stake-address build

```
( --stake-verification-key <STRING>
| --stake-verification-key-file <FILE>
| --stake-script-file <FILE>
)
(--mainnet | --testnet-magic <NATURAL>)
[--out-file <FILE>]
```

cardano-cli stake-address key-hash

```
( --stake-verification-key <STRING>
| --stake-verification-key-file <FILE>
)
[--out-file <FILE>]
```

cardano-cli stake-address registration-certificate

```
( --stake-verification-key <STRING>
| --stake-verification-key-file <FILE>
| --stake-script-file <FILE>
|--stake-address <ADDRESS>
)
--out-file <FILE>
```

cardano-cli stake-address deregistration-certificate

```
( --stake-verification-key <STRING>
| --stake-verification-key-file <FILE>
| --stake-script-file <FILE>
|--stake-address <ADDRESS>
)
--out-file <FILE>
```

cardano-cli stake-address delegation-certificate

```
( --stake-verification-key <STRING>
| --stake-verification-key-file <FILE>
| --stake-script-file <FILE>
|--stake-address <ADDRESS>
)
( --stake-pool-verification-key <STRING>
| --cold-verification-key-file <FILE>
| --stake-pool-id <STAKE-POOL-ID>
)
--out-file <FILE>
```

2 First group of options.

The first option is what will identify your stake credential. For the example we will use the stake address once again.

cardano-cli stake-address delegation-certificate

```
( --stake-verification-key <STRING>
| --stake-verification-key-file <FILE>
| --stake-script-file <FILE>
|--stake-address <ADDRESS>
)
( --stake-pool-verification-key <STRING>
| --cold-verification-key-file <FILE>
| --stake-pool-id <STAKE-POOL-ID>
)
--out-file <FILE>
```

3 And here is the difference...

This time, you will need to use your cold.vkey to serve as the Pool ID to delegate to your own pool. (Be aware that it is still possible to generate your pool ID from your cold.vkey using the command "cardano-cli stake-pool id" but the result will be the same).

cardano-cli stake-address delegation-certificate

~~(--stake-verification-key <STRING>~~
~~| --stake-verification-key-file <FILE>~~
~~| --stake-script-file <FILE>~~
|--stake-address **stake.addr**
)
~~(--stake-pool-verification-key <STRING>~~
| --cold-verification-key-file **cold.vkey**
~~| --stake-pool-id <STAKE-POOL-ID>~~
)
--out-file **delegation.cert**

4 This is the final result on your terminal.

```
user@computer:~$ cardano-cli stake-address delegation-certificate \
> --stake-address stake.addr \
> --cold-verification-key-file cold.vkey
> --out-file delegation.cert
```

Once again, from your "Air Gap" environment, you will create your draft transaction, calculate the fees, rework your draft transaction by adding the missing information and sign it. We won't change the method because it will include sensitive data like your cold.skey, your payment.skey, your stake.skey, your stake pool certificate and your delegation certificate.That being said, there is also a stake pool deposit to be made "Onchain" at the same time as the registration.

Ninth exercise: Creation of the transaction to submit your certificates

Air Gap

1 We have gathered for you all the necessary options for your transaction.

This transaction will be done to yourself in order to submit your 2 certificates "Onchain". You can repeat exercises 6 to 8 from part 1. The procedure and way of doing it are the same. Exept for "--certificate-file" option which will slightly differ.

cardano-cli transaction build-raw

```
--tx-in <TX-IN>
--tx-out <ADDRESS VALUE>
[--invalid-hereafter <SLOT>]
[--fee <LOVELACE>]
--certificate-file <CERTIFICATEFILE>
--out-file <FILE>
```

2 Your draft should look similar to this.

You set the values to 0 in the meantime to calculate the costs of the transaction you are building.

****WARNING** Make sure that your UTXO in transaction input(s) can cover the 500 ada deposit for the registration of your stake pool.**

cardano-cli transaction build-raw

```
--tx-in 4536a4d18e9dkhb34234kjbvd81e5677d#0
--tx-out $(cat paymentwithstake.addr)+0
--invalid-hereafter 0
--fee 0
--certificate-file pool.cert
--certificate-file delegation.cert
--out-file tx.raw
```

3 And now the fees calculation.

The only option value that will differ from exercise 9 of part 1 is the witness count. (the number of signing key used)

cardano-cli transaction calculate-min-fee

```
--tx-body-file <FILE>
[--mainnet | --testnet-magic <NATURAL>]
(--genesis <FILE> | --protocol-params-file <FILE>)
--tx-in-count <NATURAL>
--tx-out-count <NATURAL>
--witness-count <NATURAL>
[--byron-witness-count <NATURAL>]
```

4 But why 3 signing keys?

- cold.skey = to validate the stake pool registration certificate
- payment.skey = for the transaction itself
- stake.skey = to validate the stake credential of your wallet in each of the 2 certificates

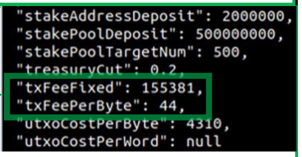
cardano-cli transaction calculate-min-fee

--tx-body-file **tx.raw**
[--**mainnet** | ~~--testnet-magic <NATURAL>~~]
(~~--genesis <FILE>~~ | --protocol-params-file **protocol.json**)
--tx-in-count **1**
--tx-out-count **1**
--witness-count **3**
~~[--byron-witness-count <NATURAL>]~~

5 What happens during the fee calculation?

The command will consider:

- The 2 certificates contained in the transaction body file.
- The number of transaction inputs
- The number of transaction outputs
- The number of signing key to be used (--witness-count)
- The parameters in the protocol parameters file that could have an influence on your transaction.



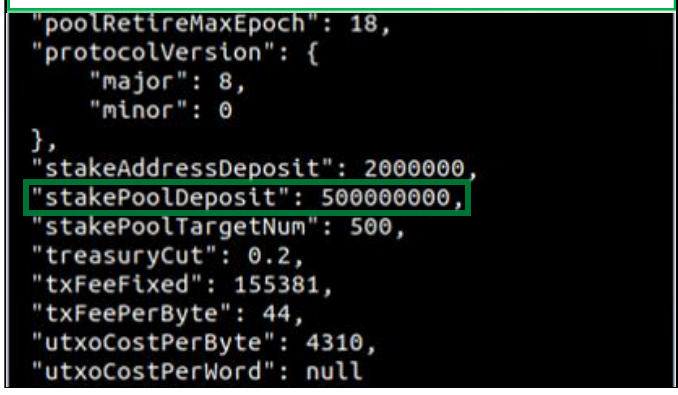
```
"stakeAddressDeposit": 2000000,
"stakePoolDeposit": 500000000,
"stakePoolTargetNum": 500,
"treasuryCut": 0.2,
"txFeeFixed": 155381,
"txFeePerByte": 44,
"utxoCostPerByte": 4310,
"utxoCostPerWord": null
```

```
user@computer:~$ cardano-cli transaction build-raw \
--tx-in 4536a4d18e9dkhb34234kjbvd81e5677d#0
--tx-out $(cat paymentwithstake.addr)+0
--invalid-hereafter 0
--fee 0
--certificate-file pool.cert
--certificate-file delegation.cert
--out-file tx.raw
```

```
user@computer:~$ cardano-cli transaction calculate-min-fee \
> --tx-body-file tx.raw \
> --mainnet \
> --protocol-params-file protocol.json \
> --tx-in-count 1 \
> --tx-out-count 1 \
> --witness-count 3
197181 Lovelace
```

6 What is stake pool deposit?

The stake pool deposit is an amount specified in the protocol.json file that each operator must deposit "onchain" in order to register their pool. This process is similar to registering a stake address.



```
"poolRetireMaxEpoch": 18,
"protocolVersion": {
    "major": 8,
    "minor": 0
},
"stakeAddressDeposit": 2000000,
"stakePoolDeposit": 500000000,
"stakePoolTargetNum": 500,
"treasuryCut": 0.2,
"txFeeFixed": 155381,
"txFeePerByte": 44,
"utxoCostPerByte": 4310,
"utxoCostPerWord": null
```

7 Now calculate your UTXO(s) minus the fees and the stake pool deposit to determine the value of --tx-out

Amount of the UTXO

Stake pool deposit

```
user@computer:~$ expr 5600000000 - 197181 – 500000000
```

Fee

```
user@computer:~$ expr 5600000000 - 197181 – 500000000
5099802819
user@computer:~$
```

8 Add the transaction fees and the output value.

For the "--invalid-hereafter" option, you must get the node's current tip and then add several minutes to the result to allow you to sign and submit your transaction before its "onchain expiration" (1 slot = 1 second).
*****Refer to exercise 6 of part 1 of this document*****

cardano-cli transaction build-raw

--tx-in 4536a4d18e9dkhb34234kjbvd81e5677d#0
--tx-out $(cat paymentwithstake.addr)+**5099802819**
--invalid-hereafter 0
--fee **197181**
--certificate-file pool.cert
--certificate-file delegation.cert
--out-file tx.raw

Add them here

9 Now all your options have their final values, you will then be ready to sign the transaction.

```
user@computer:~$ cardano-cli transaction build-raw \
--tx-in 4536a4d18e9dkhb34234kjbvd81e5677d#0 \
--tx-out $(cat paymentwithstake.addr)+5099802819 \
--invalid-hereafter 92030834 \
--fee 197181 \
--certificate-file pool.cert \
--certificate-file delegation.cert \
--out-file tx.raw
```

10 Don't forget to use your 3 signing keys as previously explained.

```
user@computer:~$ cardano-cli transaction sign \
--tx-body-file tx.raw \
--signing-key-file payment.skey \
--signing-key-file stake.skey \
--signing-key-file cold.skey \
--mainnet \
--out-file tx.signed
```

11 Transfer it and submit it on your Hot Node

```
user@computer:~$ cardano-cli transaction submit \
--mainnet \
--tx-file tx.signed
```

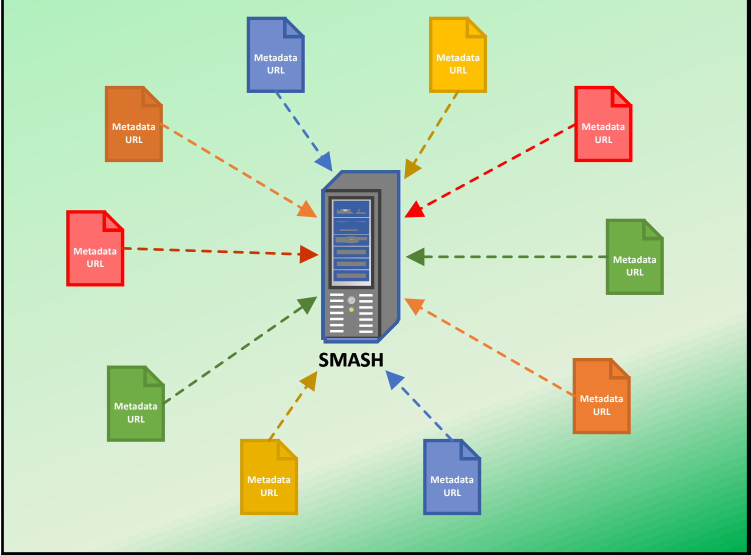
```
user@computer:~$ cardano-cli transaction submit \
> --mainnet \
> --tx-file tx.signed
transaction successfully submitted
```

Congratulation! You now have a registered stake pool. In order to ensure the validity of the metadata that you have just submitted "Onchain", we advise you to follow this next exercise "off-topic" but useful for the proper transmission of your metadata to the Cardano wallets . This exercise does not involve the cardano-cli but rather SMASH as mentioned before.

Off-topic exercise: Validating your pool metadata with SMASH

1 About SMASH...

The purpose of SMASH is to aggregate off-chain metadata that stake pools provide when they register on the Cardano blockchain.



2 What does that mean exactly?

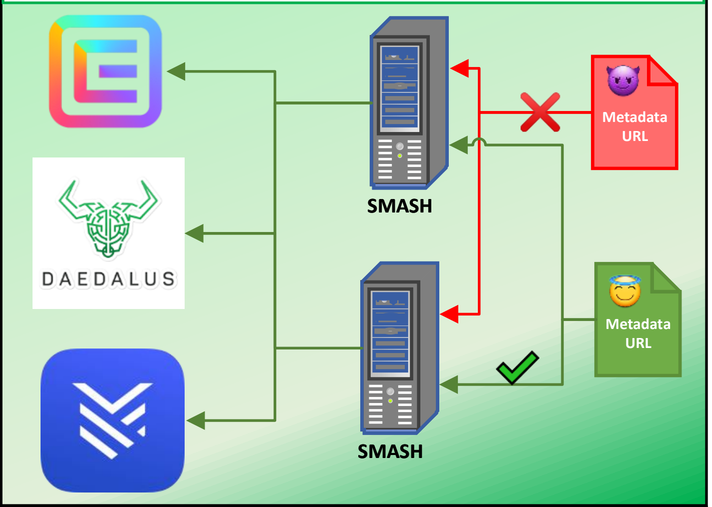
The integrity and reputation of Cardano depends on ensuring that registered stake pools are valid, that they do not duplicate ticker names or trademarks, and do not feature content that users are likely to find offensive. SMASH was designed to improve visibility on verified stake pool information for Cardano users and enable convenient navigation options

Metadata

- **owner**
- **pool name**
- **pool ticker**
- **homepage**
- **pledge address**
- **short description**

3 SMASH collects the off-chain data to make it more convenient, performant, and reliable.

The SMASH server also addresses a second consideration: the desire to moderate the content of stake pool metadata without a centralized censoring entity. More information about SMASH at: ***https://github.com/input-output-hk/cardano-db-sync/blob/master/doc/smash.md***



4 You will not build a SMASH server in this tutorial but you will use it to validate the metadata of your stake pool like this:

```
user@computer:~$ curl  "https://smash.cardano-mainnet.iohk.io/api/v1/errors/Your_pool_ID_in_HEX_format"
user@computer:~$
```

✓ If nothing happens, then that's great news. it just means that your metadata is valid. Congratulations.

5 On the other hand, if you get something like this (see below), it means that you have probably made a mistake in the writing of your metadata or submitted a metadata hash that does not match the one from your URL.**Refer to exercise 6, part 2 to re-edit your metadata.**

```
user@computer:~$ curl  "https://smash.cardano-mainnet.iohk.io/api/v1/errors/45cfc42a91bfd8f0aeb037dty546453c6f88661b5d4e0e5b94069459e345"
[{"cause":"Hash mismatch from when fetching metadata from https://testpool.github.io/poolmeta.json. Expected
622cc2ae712eec7d14c52f6c86a9abe9d9434c1295f4955dc31725b9f93ec154 but got
efa01f807d1e76b9d5d1be497e7c416568de0c0c1c9cee0316acb9b1f418e29c.","poolHash":"622cc2ae712eec7d14c52f6c86a9abe9d9434c1295f4955dc31725b9f93ec1
54","poolId":"48cfc42a91bfd8f0aeb03722e5465645654561b5d4e0e5b94069459e703","retryCount":0,"time":"29.05.2023.
17:46:54","utcTime":"1685382414.958924s"},{"cause":"URL parse error from for pool1fr8ug253hlv0pt4sxu3w6577665npkh2wpedegp55t8nsx28rkma resulted in :
InvalidUrlException \"pool1fr8ug253hlv0pt4sxu3w20665744npkh2wpedegp55t8nsx28rkma\" \"Invalid URL\
"","poolHash":"9f26210c80b9a5aee145e19d46d097e04b056b167ed6f68c93c157a9b","poolId":"48cfc42a91bfd8f0aeb03725665661b5d4e0e5b94069459e703","retryC
ount":1,"time":"29.05.2023. 17:08:32","utcTime":"1685380112.692856s"},{"cause":"Hash mismatch from when fetching metadata from https://testpool.github.io/
poolmeta.json. Expected 9f26210c80b9a5aee145e19d46d097e04b08b006f8f4b167ed6f68c93c157a9b but got
4488b447422ede85692e6ad8675d7526ac638c1148eee9fb0d172e193142afe2.","poolHash":"9f26210c80b9a5aee145e19d46d097e04b08b006f8f4b167ed6f68c93c157
a9b","poolId":"48cfc42a91bfd8f0aeb03722e53c6f88661b5d4e0e5b94069459e703","retryCount":0,"time":"29.05.2023.
16:23:36","utcTime":"1685377416.851803s"},{"cause":"URL parse error from for pool1fr8ug253hlv0pt4sxu3w20r03pnpkh2wpedegp55t8nsx28rkma resulted in :
InvalidUrlException \"pool1fr8ug253hlv0pt4sxu3w20r03pnpkh2wpedegp55t8nsx28rkma\" \"Invalid URL\
"","poolHash":"84f4e43bf074058623bcd16e7df038e936522671308fc8dc3635b54da7b82b0c","poolId":"48cfc42a91bfd8f0aeb03722e53c6f88661b5d4e0e5b94069459
e703","retryCount":1,"time":"29.05.2023. 06:42:10","utcTime":"1685342530.604564s"}]user@computer:~$
```

✗ You can read several clues from these logs(in yellow) in order to rectify the problem surrounding your metadata as quickly as possible. And when this is done and corrected, you will have to resubmit another stake pool certificate and recheck if there are new error logs with the SMASH servers

We will finish the part 2 of this tutorial with a quote from Adam Dean, a great Cardano DEV and programmer:

“Moms everywhere since forever: If your friend was jumping off a cliff, would you follow them?

-Crypto: hold my beer.”

@adamKDean