

Full Node Bootstrap

Full Node Process Epilogue

When a full node resumes network participation, it reads `consensus_seed` from `HOME/.sgx_secrets/consensus_seed.sealed`, does the same key derivation steps as above in steps 2-5.

1. Remote Attestation

The new full node verifies the remote attestation proof of the bootstrap node from `genesis.json` and creates a remote attestation proof for their own machine to show to the network that the node's Enclave is genuine.

1. Generate Registration Keypair

Using HKDF-SHA256, `hkdf_salt` and `nonce` (a 256 bit true random) a private key is derived. From `registration_privkey` calculate `registration_pubkey`

...

Copy `registration_privkey=hkdf({ salt:hkdf_salt, ikm:Nonce-256bit, }); // 256 bits`

`registration_privkey=calculate_curve25519_pubkey(registration_pubkey);`

...

1. Authorize Full Node

The node needs to send `ansecretcli tx register auth` transaction with the following inputs:

- The remote attestation proof the node's Enclave is genuine
- `registration_pubkey`
- 256 bits true random `nonce`
-

1. Network Receives Auth

On the consensus layer, inside the enclave of every full node the auth transaction comes in.

The Network validator nodes:

- Receive the `secretcli tx register auth`
- transaction
- Verify the remote attestation proof that the new node's Enclave is genuine
-

1. Generate `seed_exchange_key`

`seed_exchange_key` : An [AES-128-SIV](#) encryption key is used to send `consensus_seed` to the new node

This key is derived in several steps:

- first `seed_exchange_ikm`
- is derived using [ECDH](#)
- ([x25519](#)
-) with `consensus_seed_exchange_privkey`
- (which exists in the enclave) and `registration_pubkey`
- (which is a public identifier of the full node)
-

This IKM is never publicly available and protects the Secret network private entropy

...

Copy `seed_exchange_ikm = ecdh({ privkey: consensus_seed_exchange_privkey, pubkey: registration_pubkey, }); // 256 bits`

...

In the second step `seed_exchange_key` is derived using HKDF-SHA256 from `seed_exchange_ikm` and `nonce`. When sending the `seed_exchange_key` to new nodes the Nonce is added as plaintext, it just serves the function of making each `seed_exchange_key` unique.

...

```
Copy seed_exchange_key = hkdf({ salt: hkdf_salt, ikm: concat(seed_exchange_ikm, nonce), }); // 256 bits
```

...

1. Sharing consensus_seed with the new node

The seed_exchange_key generated in step 5 is used to encrypt the consensus_seed. The AD for this encryption algorithm is the public key of the new node: new_node_public_key. All this logic is done inside the Authorization transaction. secretcli tx register auth outputs the encrypted_consensus_seed

...

```
Copy encrypted_consensus_seed = aes_128_siv_encrypt({ key: seed_exchange_key, data: consensus_seed,
ad: new_node_public_key, });
```

```
return encrypted_consensus_seed;
```

...

The encrypted output is received by the new full node. The new node now has access to the encrypted_consensus_seed and will have to decrypt to plaintext to receive the consensus_seed

1. New full node generates own seed_exchange_key

The AES-128-SIV encryption key seed_exchange_key is used to decrypt consensus_seed. To derive this the reverse logic is followed highlighted in step 5.

First the same seed_exchange_ikm is derived using [ECDH \(x25519\)](#) with consensus_seed_exchange_pubkey (public in genesis.json) and registration_privkey (available only inside the new node's Enclave). This is the DH-key exchange in action as this is the reverse public/private input of the IKM generation in step 5.

...

```
Copy seed_exchange_ikm = ecdh({ privkey: registration_privkey, pubkey: consensus_seed_exchange_pubkey, // from
genesis.json }); // 256 bits
```

...

seed_exchange_key is derived using HKDF-SHA256 with seed_exchange_ikm and nonce

...

```
Copy seed_exchange_key = hkdf({ salt: hkdf_salt, ikm: concat(seed_exchange_ikm, nonce), }); // 256 bits
```

...

1. Decrypting encrypted_consensus_seed

encrypted_consensus_seed is encrypted with AES-128-SIV, seed_exchange_key as the encryption key and the public key of the registering node as the AD as the decryption additional data. The new node now has all of these parameters inside its Enclave, so it's able to decrypt consensus_seed from encrypted_consensus_seed and then seal consensus_seed to disk at HOME/.sgx_secrets/consensus_seed.sealed

...

```
Copy consensus_seed = aes_128_siv_decrypt({ key: seed_exchange_key, data: encrypted_consensus_seed,
ad: new_node_public_key, });
```

```
seal({ key: "MRSIGNER", data: consensus_seed, to_file: "HOME/.sgx_secrets/consensus_seed.sealed", });
```

''' Last updated 1 year ago On this page Was this helpful? [Edit on GitHub](#) [Export as PDF](#)