Filecoin-PoRep and PoSt algorithm source code guide

FileCoin (https://learnblockchain.cn/tags/FileCoin)

Filecoin-PoRep and PoSt algorithm source code guide

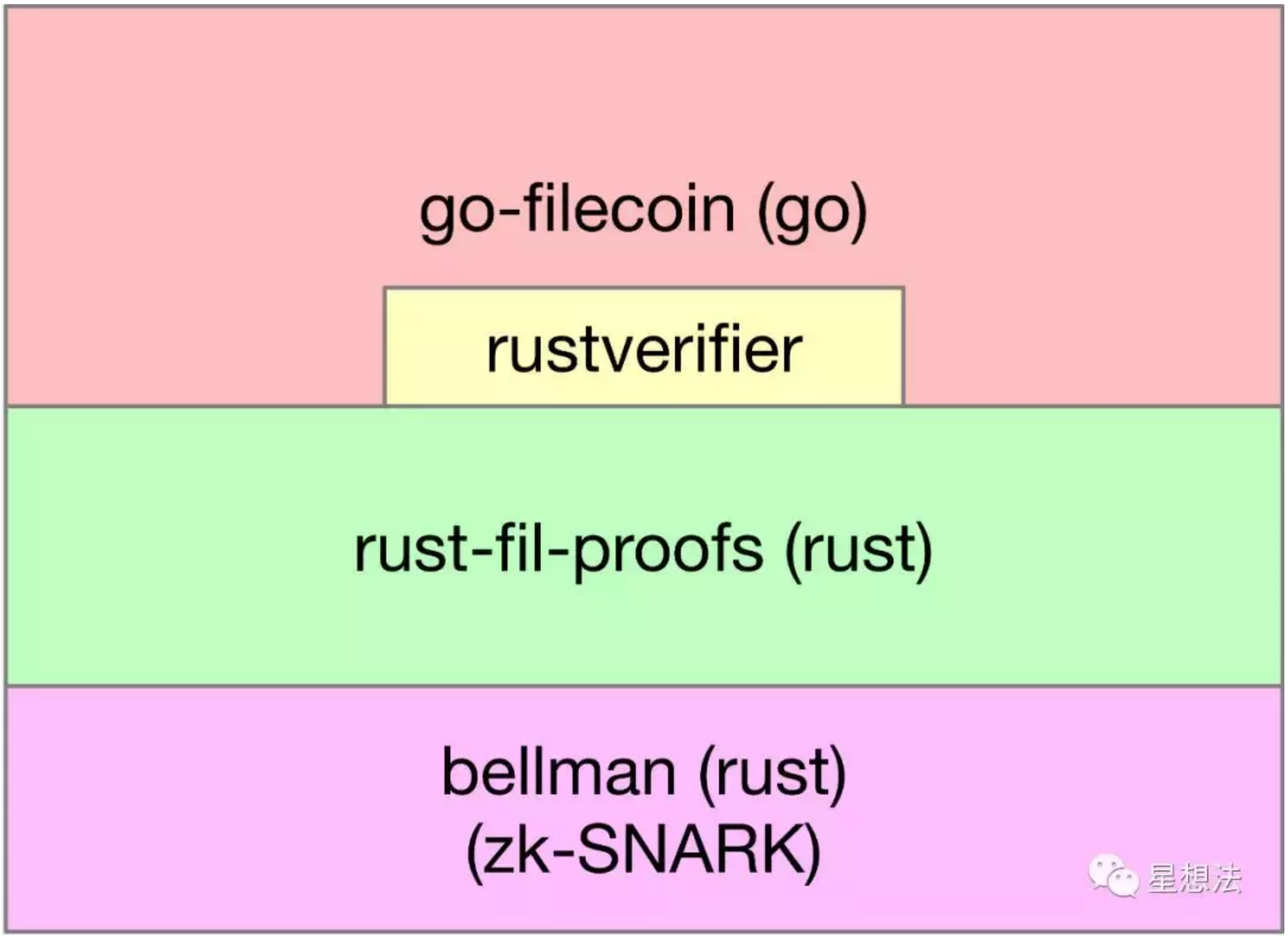
" *Entrepreneurship is a very hard process. It is a process of breaking up oneself and reconnecting resources. Working in a company depends on the ability to connect and integrate company resources. Entrepreneurship, connected social resources, open resources. Uncertainty, self-denial, self-affirmation, interleaving. How to quickly determine and connect resources also has a methodology. Last weekend, I went to Fragrant Hills with my good friends and took a class on entrepreneurship theory. I gave initiation. Let me share with you some terms: class splitting, psychological motivation, and project management. Good friends give lectures to others, 10,000 yuan an hour. When I have the opportunity in the future, I will talk about my experience and understanding slowly.* 』

A few weeks ago, I looked at Filecoin's code and sorted out some of Filecoin's concepts, architecture and protocol- Filecoin logic combing and source code guide (https://learnblockchain.cn/article/679) .

The data storage proof of PoRep and PoSt is realized through FPS module. The entire FPS module is implemented through Rust language. Technicians always want to break the casserole and ask to the end, what is the proof process of PoRep/PoSt? From the perspective of the source code, I will talk about the logic call part of the data proof. It is convenient for other friends to understand or optimize related logic. The derivation code of zero-knowledge proof will be introduced in detail in subsequent articles.

01 Filecoin code module dependencies

From the perspective of PoRep and PoSt storage proof, the Filecoin code module consists of the following three layers:



rust-fil-proofs is a concrete realization of storage proofs, implemented by Rust language. The rust-fil-proofs module relies on the bellman project (zero-knowledge proof). The bellman project is a zero-knowledge proof module used by the ZCash project. The bellman project is also implemented in Rust language. In the rustverifier.go file of go-filecoin, the call from go to the rust language is implemented.

In this guide, the last commit information of the go-filecoin code is as follows:

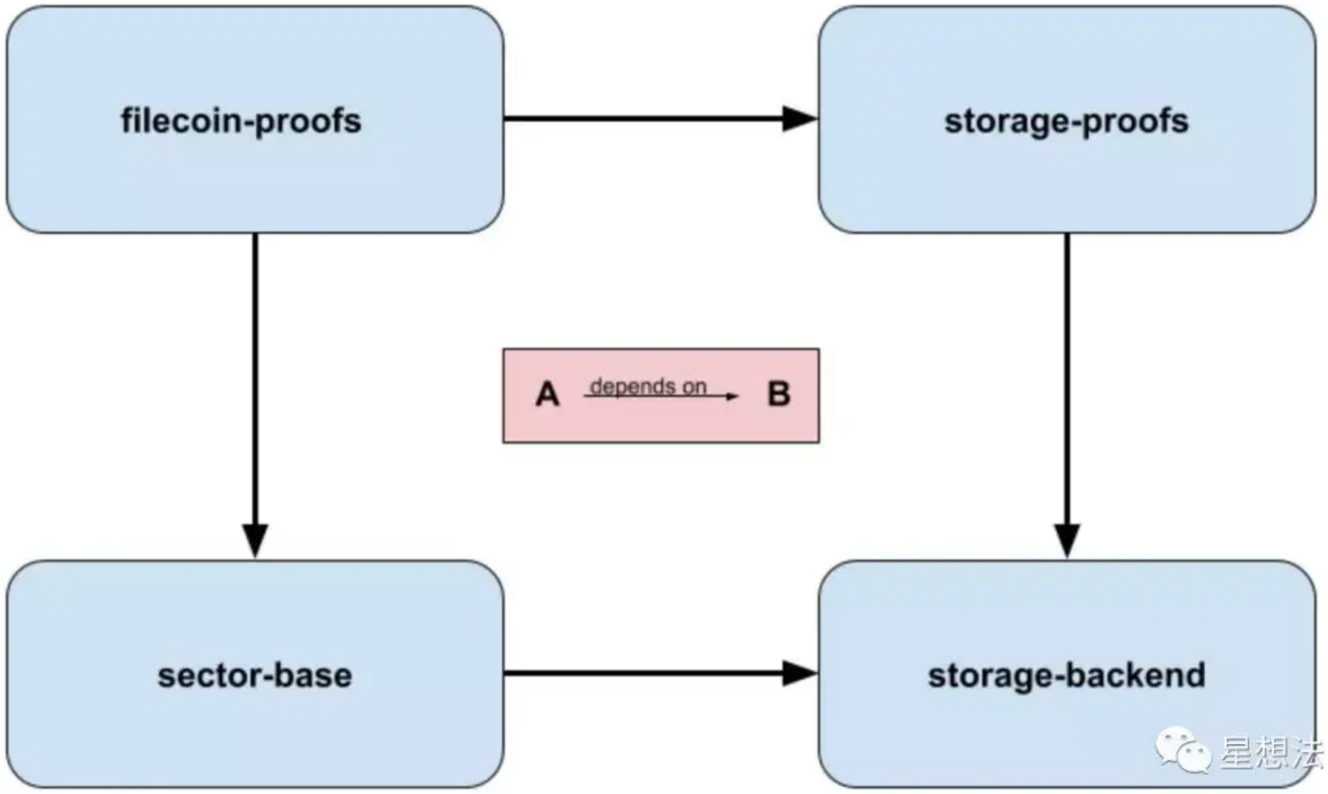
```
commit b716be02c0e15436141a5c20274a38aec749490e (HEAD -> master, tag: nightly-13394-b716be, origi

Author: Sidney Keese sidke.z@gmail.com

Date:   Wed Mar 27 14:06:13 2019 -0700

    use precompiled bls-signatures library (#2368)
```

The last commit information of the rust-fil-proofs code is as follows:

```
commit 808dc884642acbf7b78b282eb7d933c7cc2cc3a7 (HEAD -> master, origin/master, origin/HEAD)

Author: wayne yang wayne.w.yang@foxmail.com

Date:   Mon Mar 25 21:12:40 2019 +0800

    docs: delete the outdated link
```

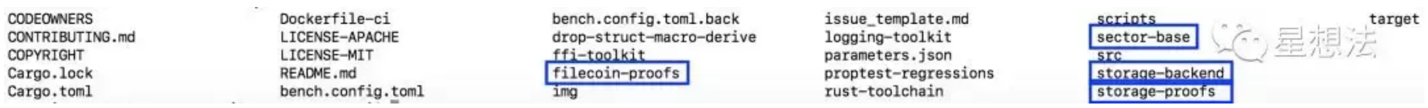
02 The framework of rust-fil-proofs

The rust-fil-proofs function mainly consists of four sub-modules:



filecoin-proofs implements the filecoin storage proof interface and relies on two other modules: storage-proofs (the logic of storage proofs) and sector-base (the interface for data storage). These two modules in turn rely on the storage-backend module to achieve data storage.

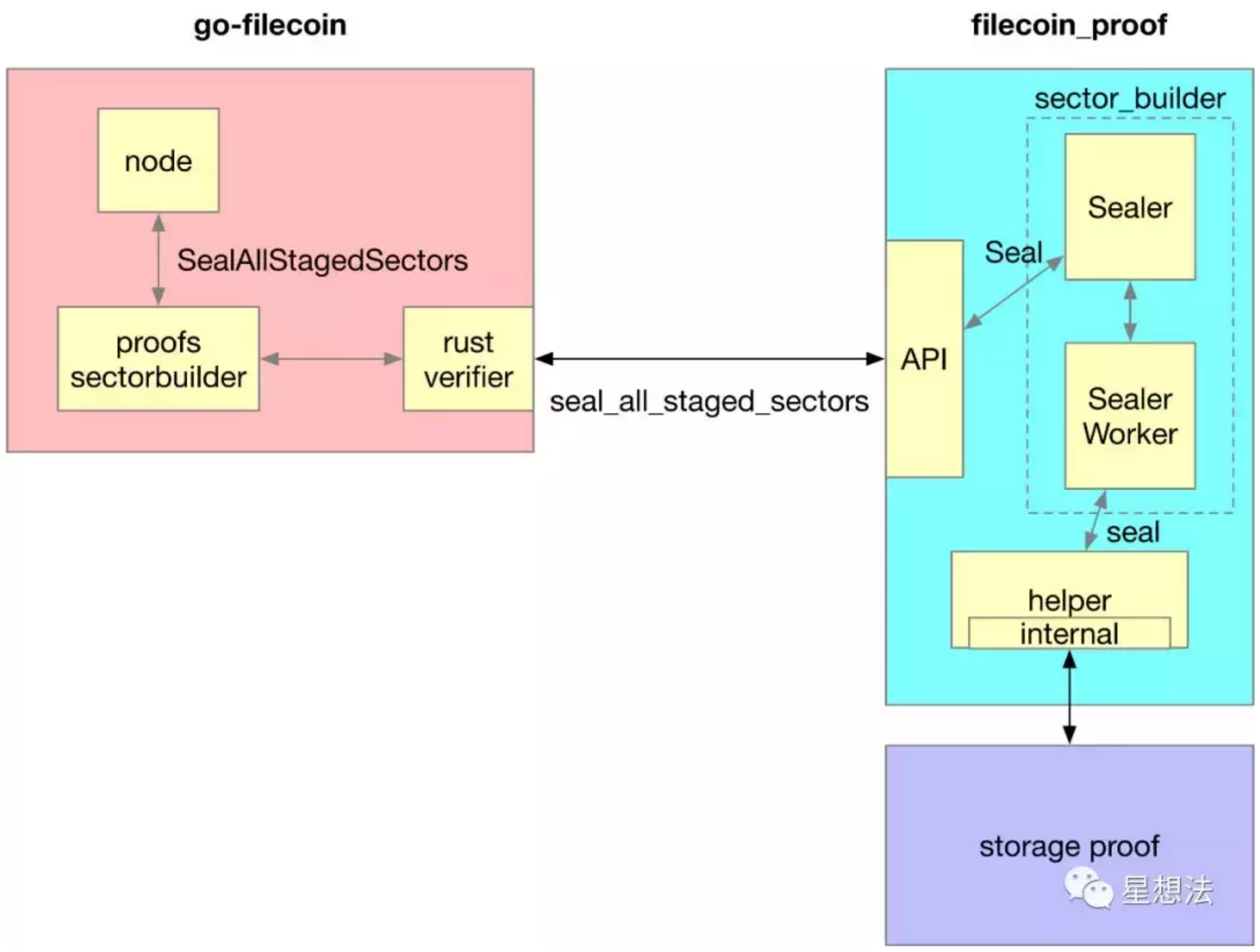
The relevant code is in these four directories:



```
CODEOWNERS          Dockerfile-ci       bench.config.toml.back     issue_template.md       scripts           target
CONTRIBUTING.md     LICENSE-APACHE      drop-struct-macro-derive   logging-toolkit         sector-base
COPYRIGHT           LICENSE-MIT         ffi-toolkit                parameters.json         src
Cargo.lock          README.md           filecoin-proofs            proptest-regressions    storage-backend
Cargo.toml          bench.config.toml   img                        rust-toolchain          storage-proofs
```

03 PoRep generation and verification logic

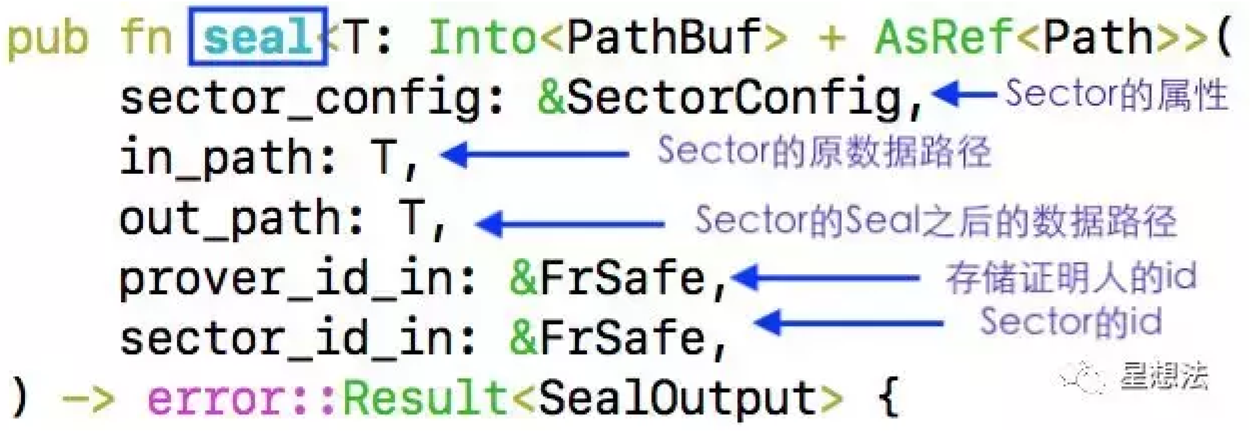
PoRep is Proof of Replicate, proof of data storage. The calling relationship between modules is as follows:



When the node.go code starts mining, it calls the SealAllStagedSectors function once every 120 blocks (Seal all the data of the Staged Sector). By the way, the user's data is stored in Sectors (currently set to 256M). Sector has three states: Staging (data has not been full and has not timed out), Staged (data is full or timed out), Sealed (data has been sealed and stored).

After the filecoin-proof module receives the request, it confirms all the current Staged Sectors, and seals each Sector.

The core logic is in the seal function of the rust-fil-proofs/filecoin-proofs/src/api/internal.rs code. The prototype of the seal function is as follows:

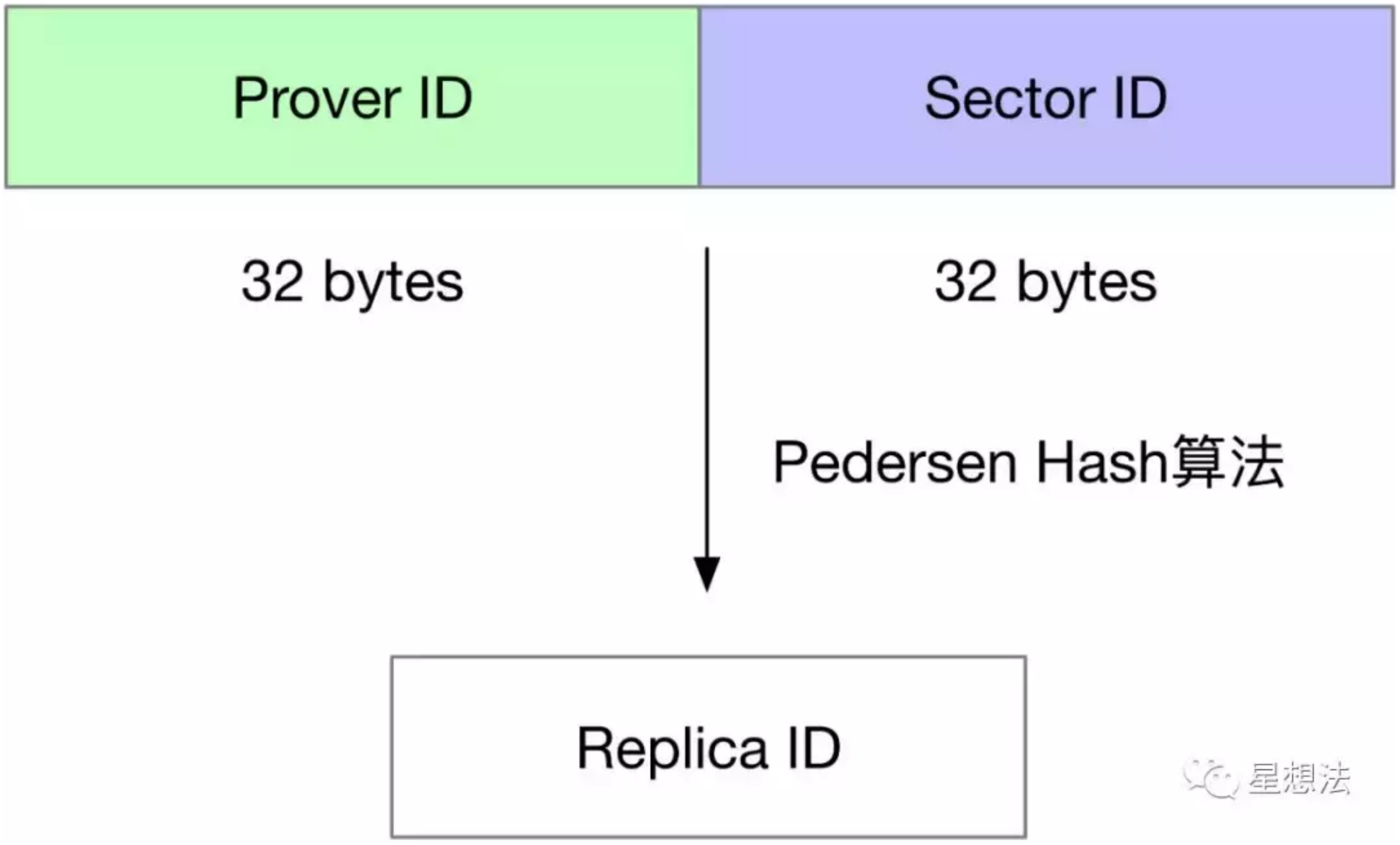


```
pub fn seal<T: Into<PathBuf> + AsRef<Path>>(
    sector_config: &SectorConfig,
    in_path: T,
    out_path: T,
    prover_id_in: &FrSafe,
    sector_id_in: &FrSafe,
) -> error::Result<SealOutput> {
```

The seal function copies the original data of in_path and stores it in out_path. In the sealing process, the attributes of the sector, the id of the certifier and the id of the sector are provided.

01 Replica ID

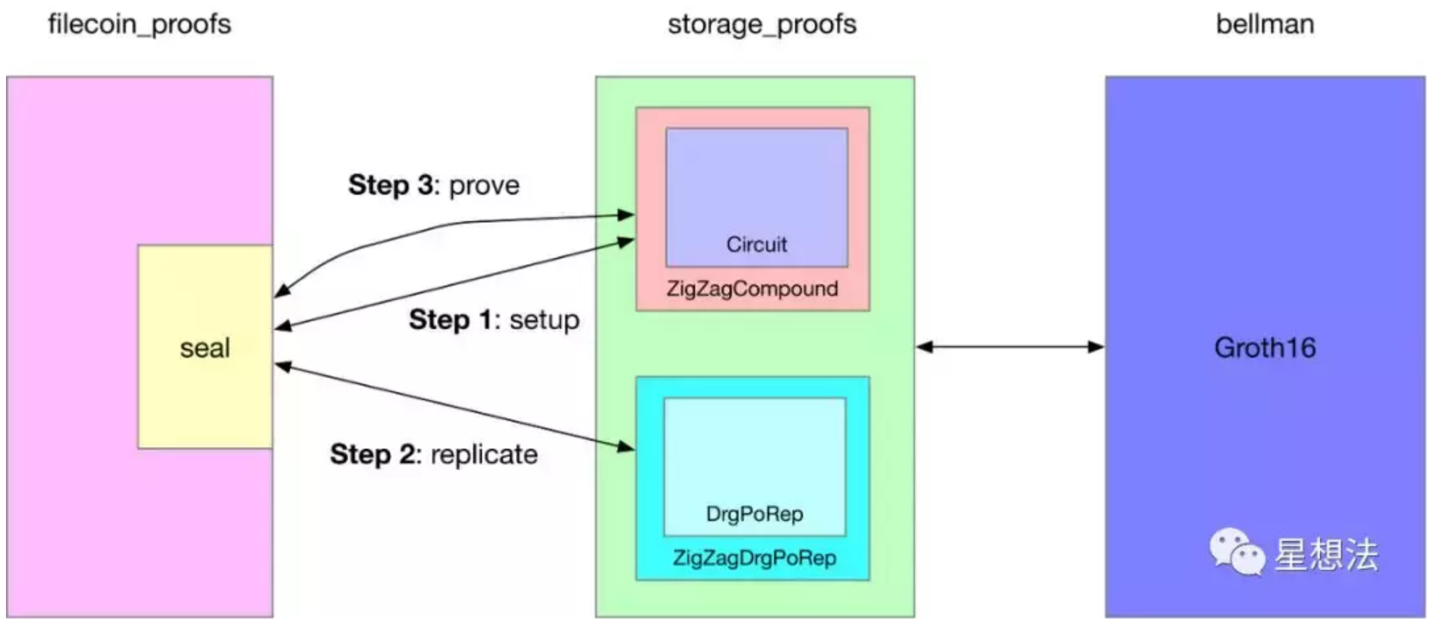
For each Sector that has been sealed, set a Replica ID. Replica ID is a hash value, and the calculation process is as follows:



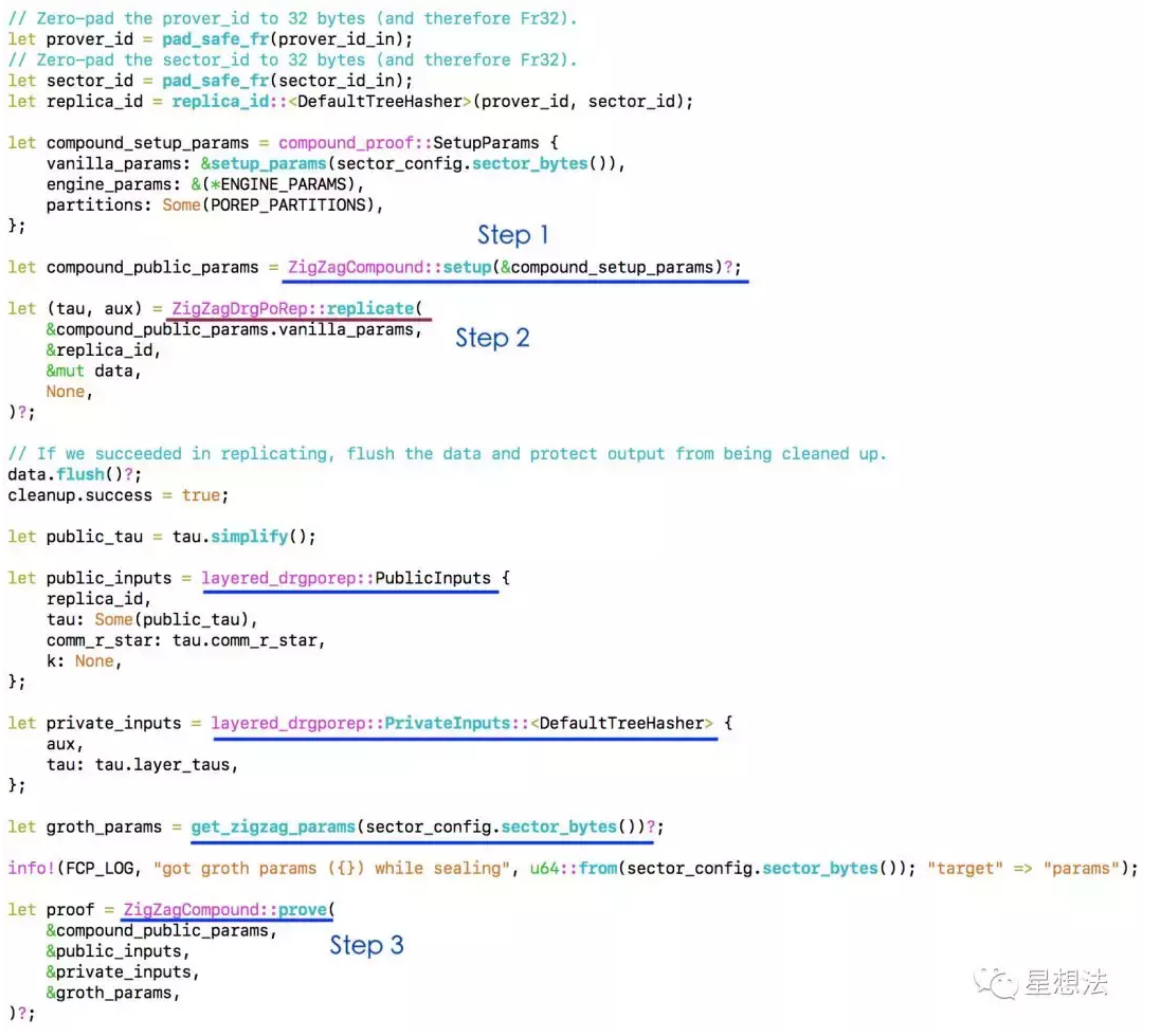
The corresponding implementation is the replica_id function of rust-fil-proofs/storage-proofs/src/porep.rs.

02 Overall logic

The generation logic of PoRep includes two parts: 1/data replication (replicate) 2/data replication proof. Step1/Step3 realizes the proof of copying, and Step2 realizes the copying of data.



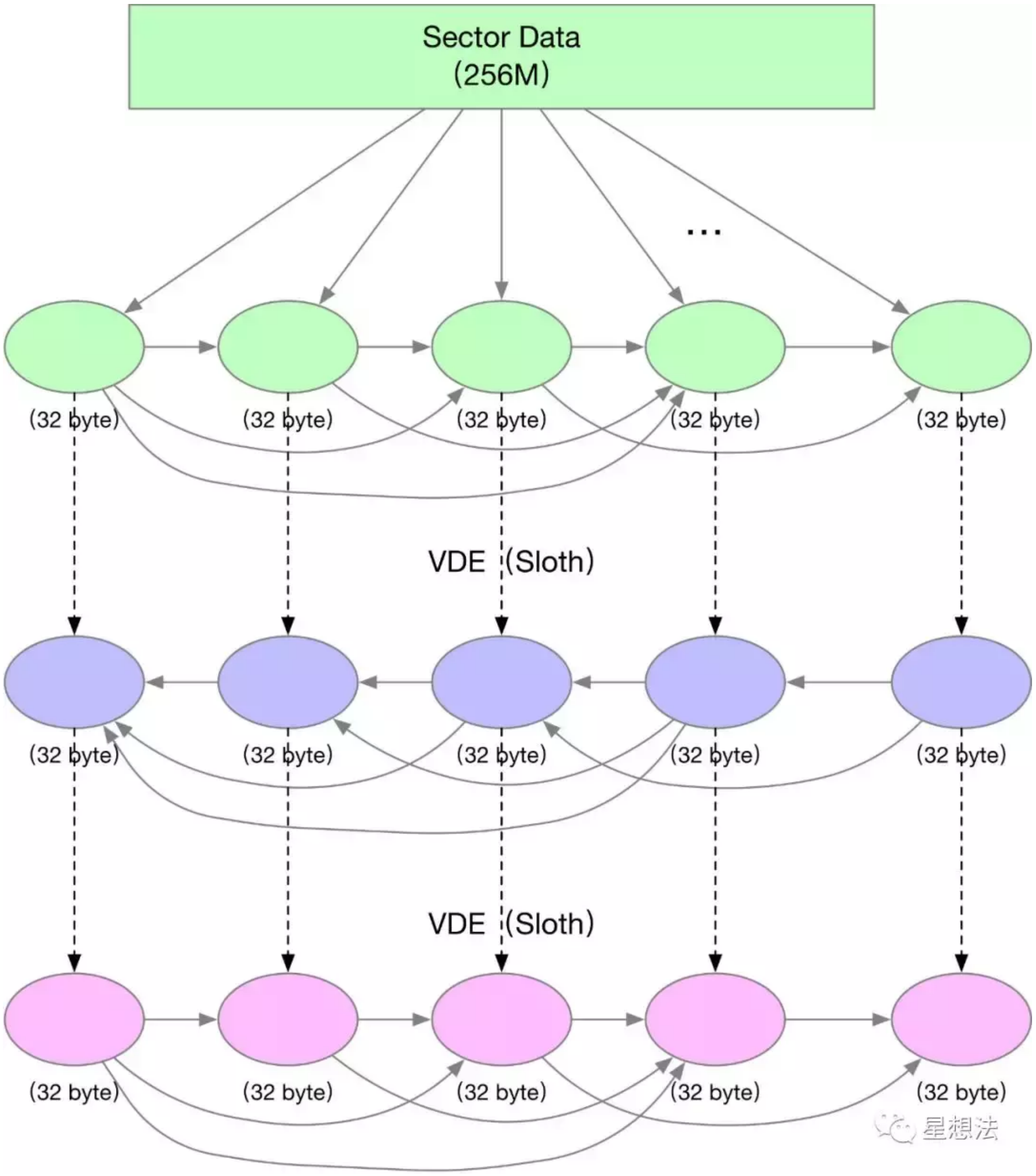
Post a piece of code to facilitate comparison and understanding of friends:



```
// Zero-pad the prover_id to 32 bytes (and therefore Fr32).
let prover_id = pad_safe_fr(prover_id_in);
// Zero-pad the sector_id to 32 bytes (and therefore Fr32).
let sector_id = pad_safe_fr(sector_id_in);
let replica_id = replica_id::<DefaultTreeHasher>(prover_id, sector_id);

let compound_setup_params = compound_proof::SetupParams {
    vanilla_params: &setup_params(sector_config.sector_bytes()),
    engine_params: &(*ENGINE_PARAMS),
    partitions: Some(POREP_PARTITIONS),
};

let compound_public_params = ZigZagCompound::setup(&compound_setup_params)?;

let (tau, aux) = ZigZagDrgPoRep::replicate(
    &compound_public_params.vanilla_params,
    &replica_id,
    &mut data,
    None,
)?;

// If we succeeded in replicating, flush the data and protect output from being cleaned up.
data.flush()?;
cleanup.success = true;

let public_tau = tau.simplify();

let public_inputs = layered_drgporep::PublicInputs {
    replica_id,
    tau: Some(public_tau),
    comm_r_star: tau.comm_r_star,
    k: None,
};

let private_inputs = layered_drgporep::PrivateInputs::<DefaultTreeHasher> {
    aux,
    tau: tau.layer_taus,
};

let groth_params = get_zigzag_params(sector_config.sector_bytes())?;

info!(FCP_LOG, "got groth params ({}) while sealing", u64::from(sector_config.sector_bytes()); "target" => "params");

let proof = ZigZagCompound::prove(
    &compound_public_params,
    &public_inputs,
    &private_inputs,
    &groth_params,
)?;
```

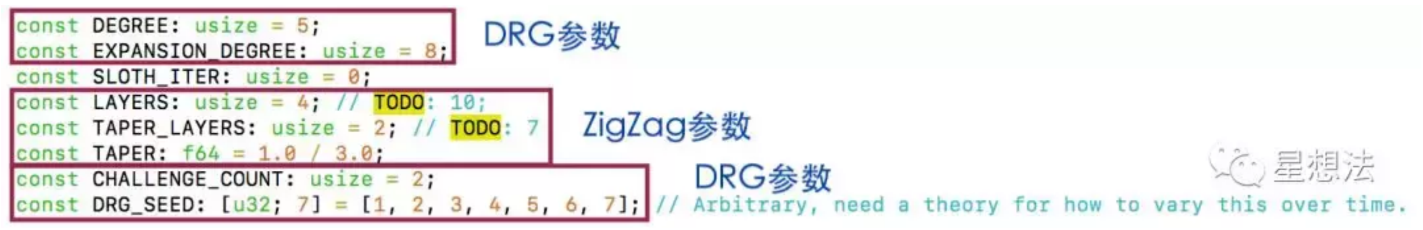
03 Data replication logic

The full name of PoRep algorithm is ZigZag-DRG-PoRep. The overall process is as follows:



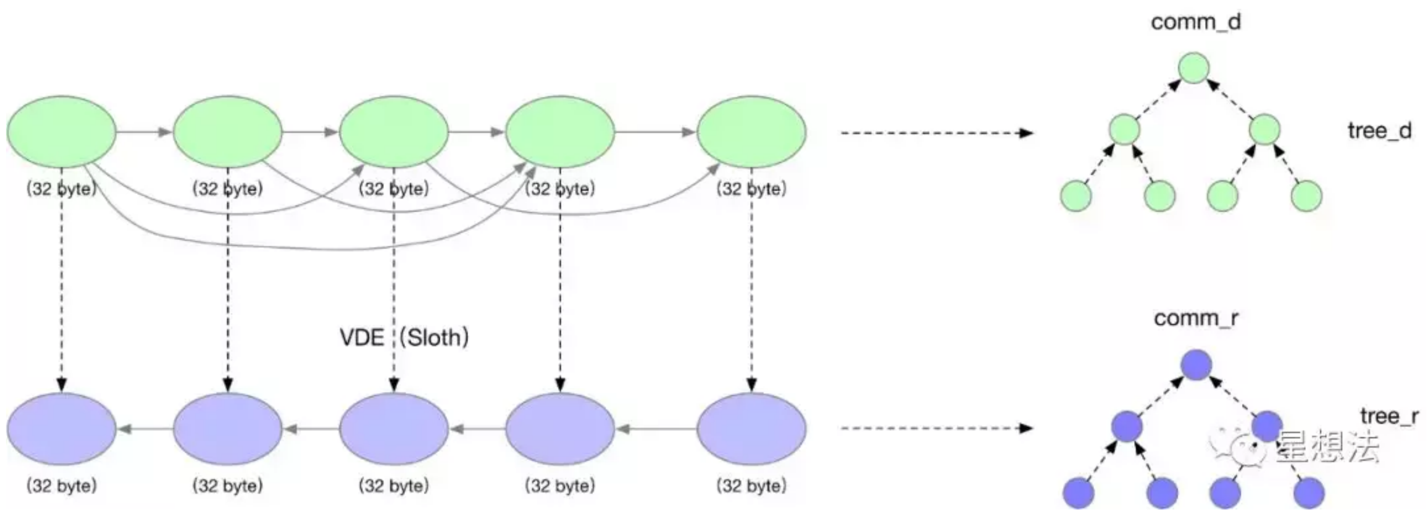
The unsealed raw data in the sector is first divided into small data one by one, each small data 32 bytes. These small data are connected according to DRG (Depth Robust Graph). According to the dependency of each small data, through the VDE (Verifiable Delay Encode) function, calculate all the small data of the next layer. The entire PoRep calculation process is divided into several layers (the current code is set to 4 layers), carefully observe the arrow direction of the DRG relationship of each layer, the upper layer is to the right, and the next layer is to the left, hence the name ZigZag (Z word type).

View the parameters set by VDE and ZigZag in rust-fil-proofs/filecoin-proofs/src/api/internal.rs:



```
const DEGREE: usize = 5;
const EXPANSION_DEGREE: usize = 8;
const SLOTH_ITER: usize = 0;
const LAYERS: usize = 4; // TODO: 10;
const TAPER_LAYERS: usize = 2; // TODO: 7
const TAPER: f64 = 1.0 / 3.0;
const CHALLENGE_COUNT: usize = 2;
const DRG_SEED: [u32; 7] = [1, 2, 3, 4, 5, 6, 7]; // Arbitrary, need a theory for how to vary this over time.
```

The calculation logic of each layer is as follows:



The input of each layer is called d (data), and the result of VDE of each layer is called r (replica). For each level of input, construct a Merkel tree, the root of which is comm_d, and the data structure of the entire tree is called tree_d. For the output of each layer, construct a Merkel tree, the root of the tree is comm_r, and the data structure of the entire tree is called tree_r.

Introduce two more terms: TAU, AUX.

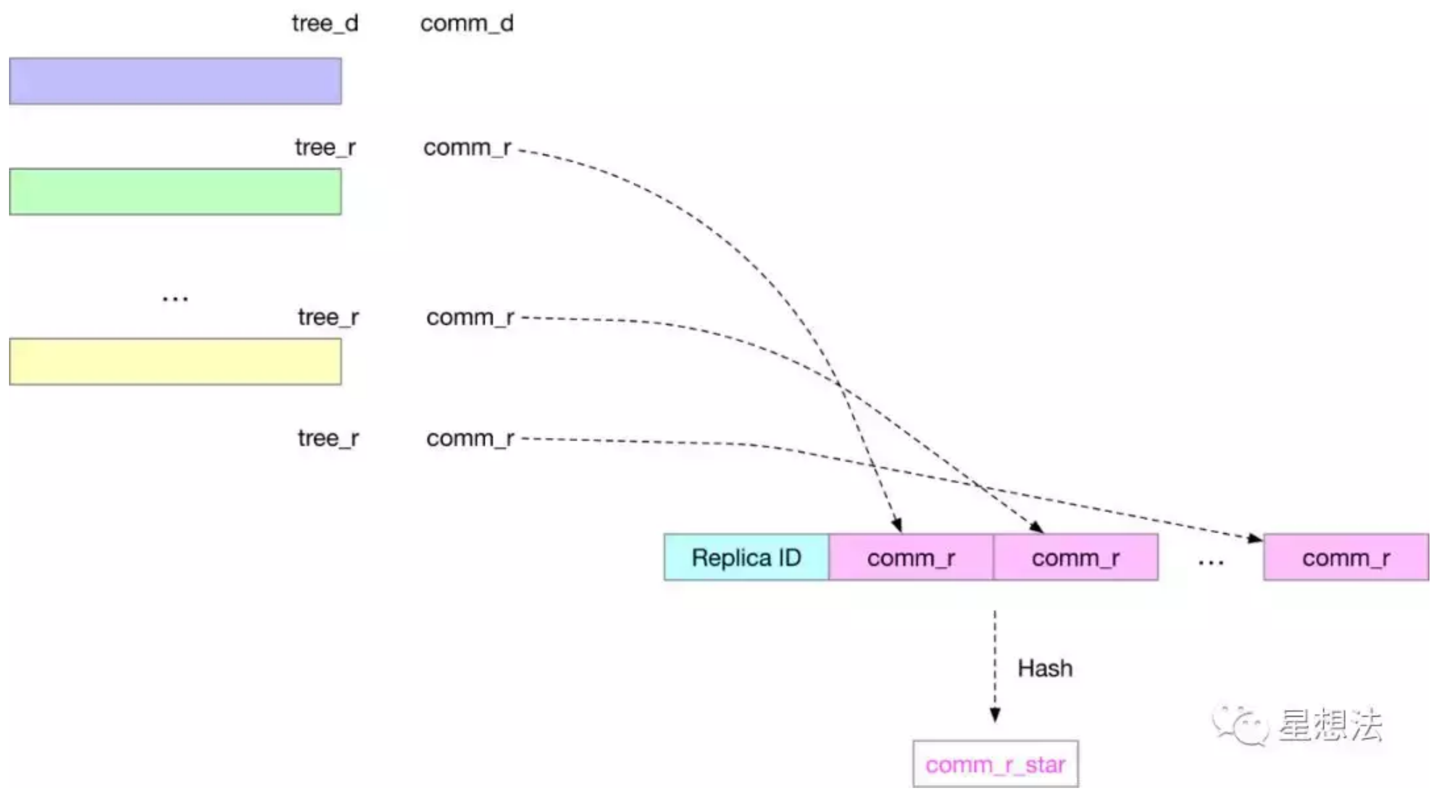
TAU: Greek letters, the roots of one or more Merkle trees are called TAU.

$\mathrm{T}\tau$

AUX: Short for Auxiliary, the structure of one or more Merkle trees is called AUX.

For a layer of replica, TAU includes comm_d and comm_r, and AUX includes tree_d and tree_r.

For the entire PoRep, that is, the multi-layer replica, TAU and AUX are shown as follows:



The comm_r_star is the result of the hash of the comm_r data and replica id data of each layer.

For the specific implementation of the source code, please see two functions:

1. `rust-fil-proofs/storage-proofs/src/layered_drgporep.rs` The replicate function of Layers in

2. `rust-fil-proofs/storage-proofs/src/drgporep.rs` The replicate function of PoRep in

04 Copy proof logic

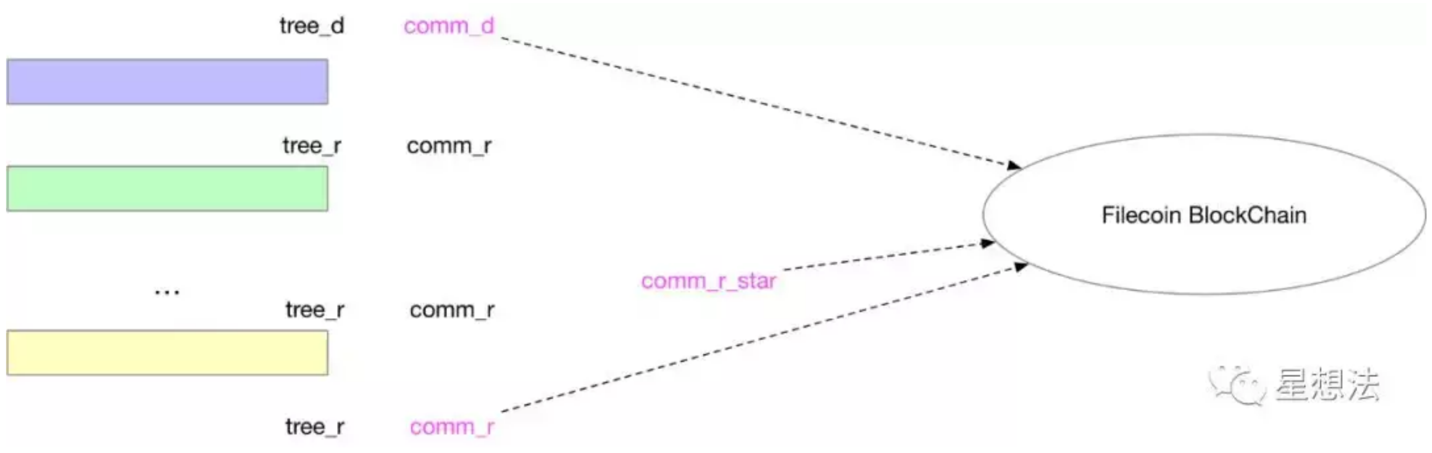
After copying the data, a zero-knowledge proof needs to be provided. The proof logic requires a lot of space to be introduced carefully, involving QAP, KCA, Groth16, homomorphic hiding, and bilinear mapping. A follow-up article will explain the relevant logic in detail. This article first has some concepts: there are two steps required to generate a proof (Proof): setup and prove.

setup: Set the parameters of the certification

Prove: Provide the content that needs to be proved, including public data (public inputs), private data (private inputs) and Groth parameters. As can be seen in the code in 2.b, The public data certified by PoRep includes comm_d of the original data, comm_r and comm_r_star of the last layer.

The private data certified by PoRep includes comm_r/comm_d of all layers and tree_r/tree_d of each layer.

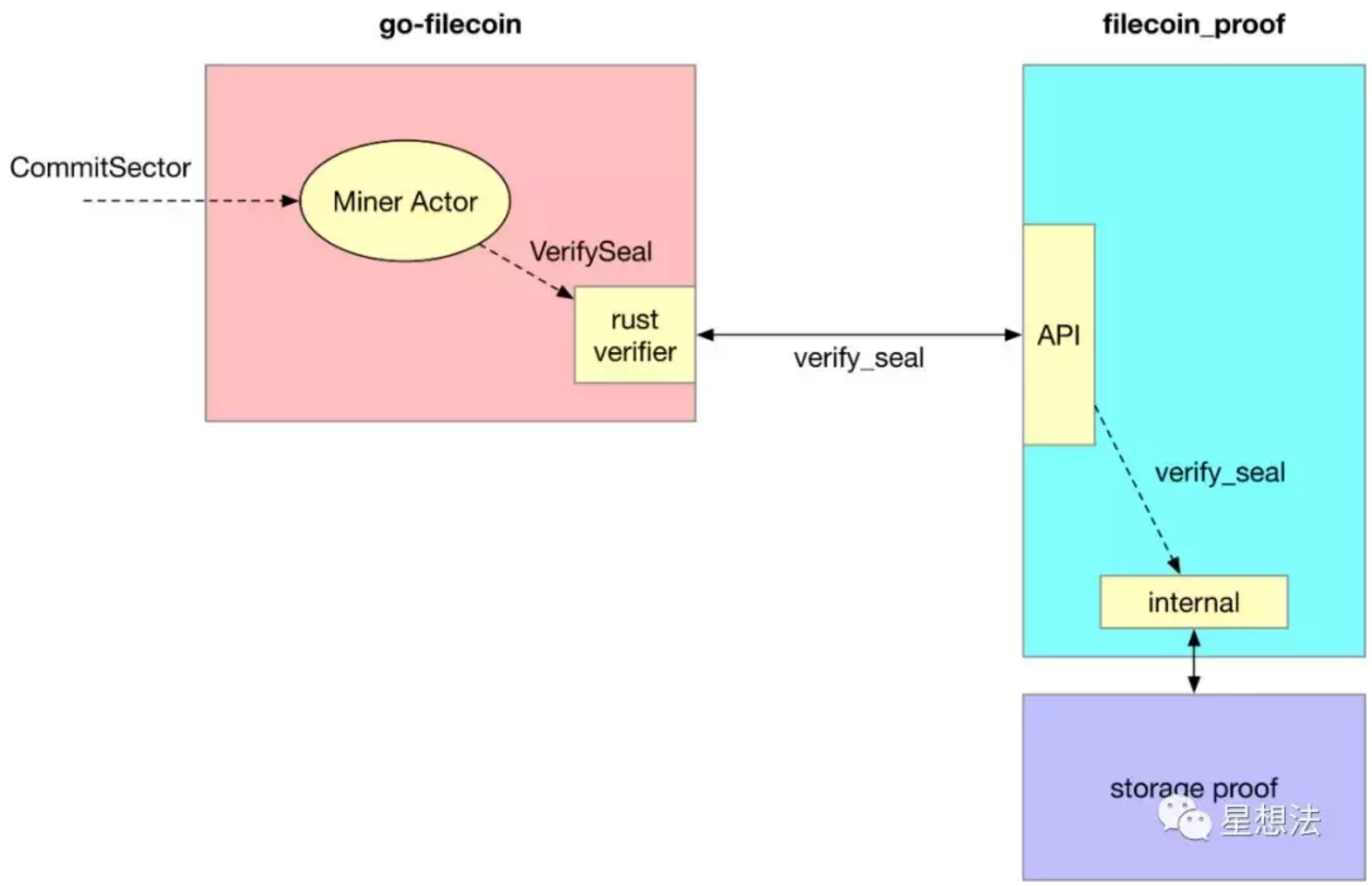
It is worth mentioning that the data that PoRep submits to the Filecoin blockchain is: PoRep's public data and proof of replication data.



The entire process of data copying takes a long time. Currently, 1G of data takes 40-50 minutes.

05 Copy verification logic

After a storage miner generates PoRep, it submits the proof information to the blockchain through the CommitSector interface. When all miners of the Filecoin blockchain receive CommitSector's Message transaction, they call VerifySeal for copy verification. The general process is as follows:



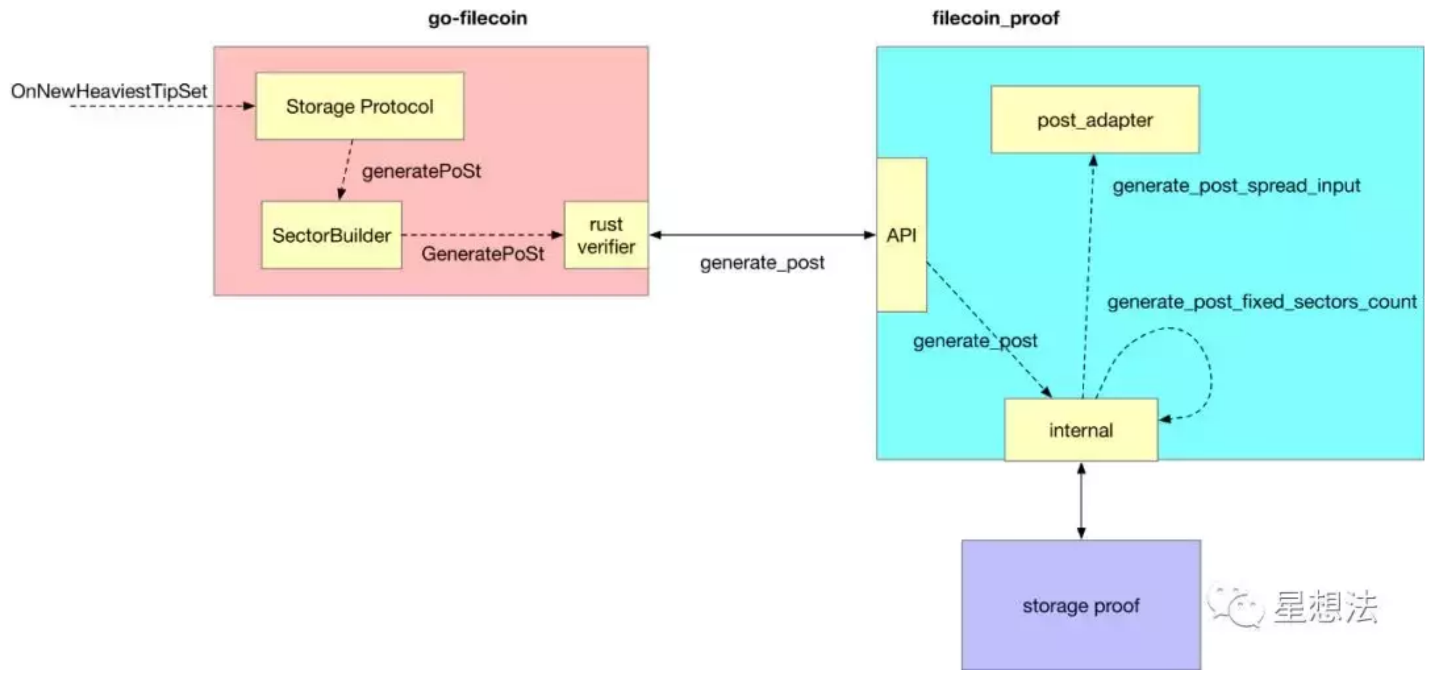
For the specific implementation of the source code, please view `rust-fil-proofs/storage-proofs/src/api/internal.rs` the `verify_seal` function.

04 PoSt generation and verification logic

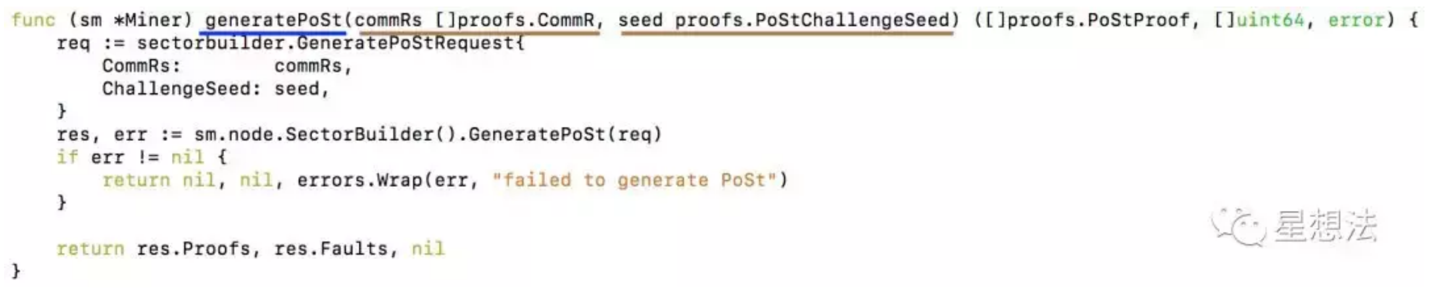
After a storage miner stores (Seal) user data, every 20,000 blocks, a proof of PoSt (Proof of Space Time) must be provided, that is, a proof that a storage miner still has user data. With 20,000 blocks, each block is 30 seconds, which means that a proof is submitted every 6 days.

01 PoSt generation logic

The generation of PoSt is based on the most recent storage state, so it is best to submit proof of PoSt within one block (30 seconds). The calling relationship of PoSt generation logic is as follows:

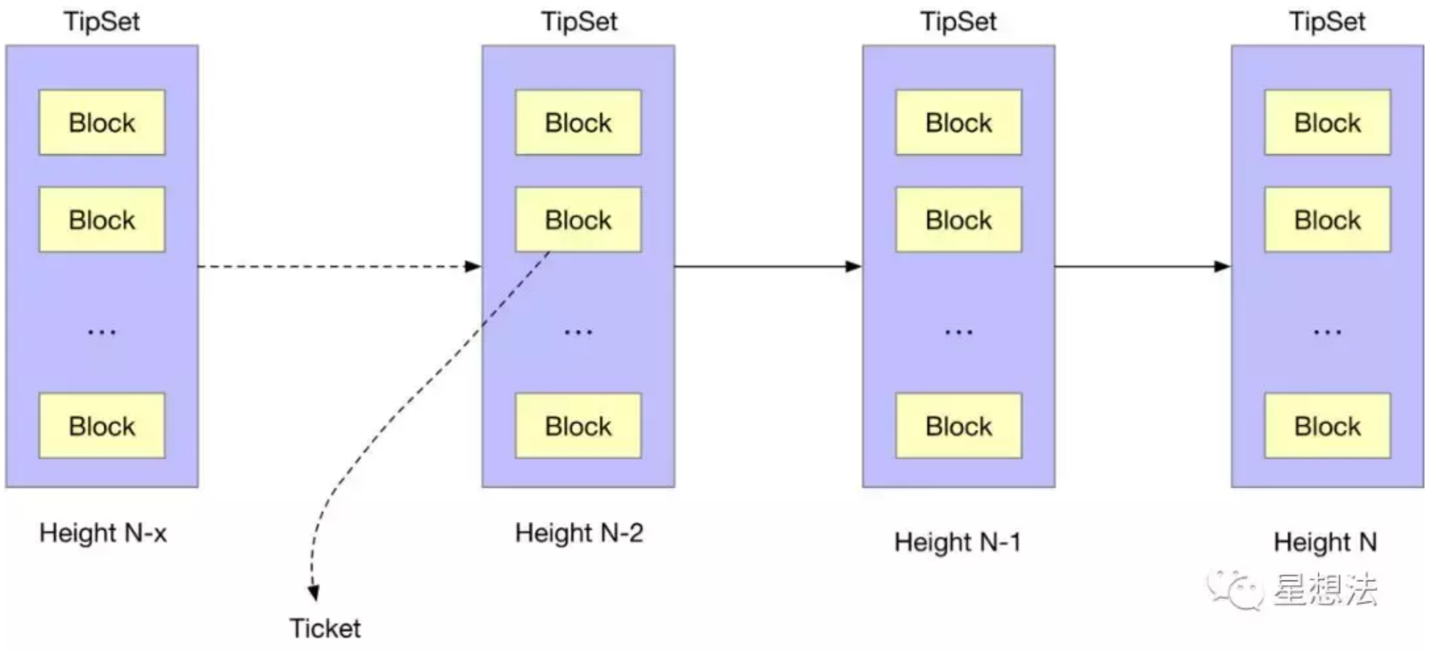


When a new block is generated (OnNewHeaviestTipSet function), each miner who provides storage checks whether it needs to provide proof of PoSt. If necessary, call the generatePoSt function to generate a certificate.



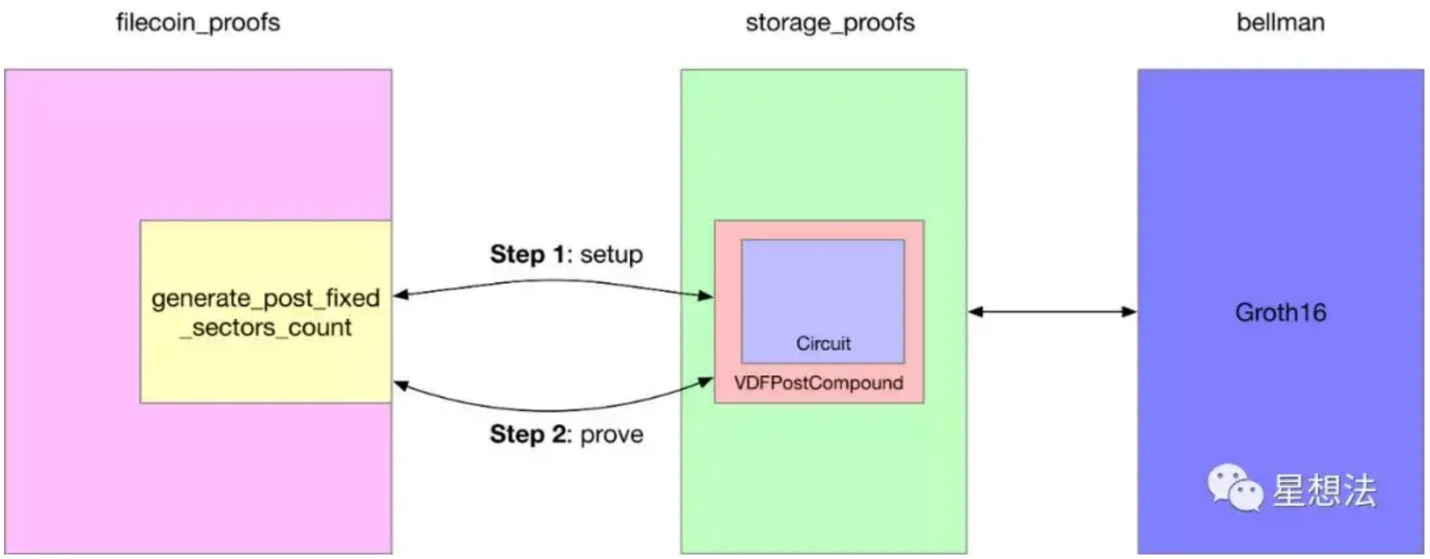
```
func (sm *Miner) generatePoSt(commRs []proofs.CommR, seed proofs.PoStChallengeSeed) ([]proofs.PoStProof, []uint64, error) {
	req := sectorbuilder.GeneratePoStRequest{
		CommRs:        commRs,
		ChallengeSeed: seed,
	}
	res, err := sm.node.SectorBuilder().GeneratePoSt(req)
	if err != nil {
		return nil, nil, errors.Wrap(err, "failed to generate PoSt")
	}

	return res.Proofs, res.Faults, nil
}
```

The generatePoSt function provides two parameters: all comm_r information submitted to the blockchain and random challenge information (ChallengeSeed). All comm_r information submitted to the blockchain can be obtained by querying the blockchain. ChallengeSeed is generated by the currentProvingPeriodPoStChallengeSeed function, and the generation logic is as follows:



Simply put, look forward to several TipSets from the current block height to find the smallest Ticket among the many blocks in a certain TipSet. The Ticket is ChallengeSeed.

In the latest code logic, PoSt generates a Proof for every two Sectors, which is the logic implemented by post_adapter. Submitted to the blockchain is a list of many Proofs.

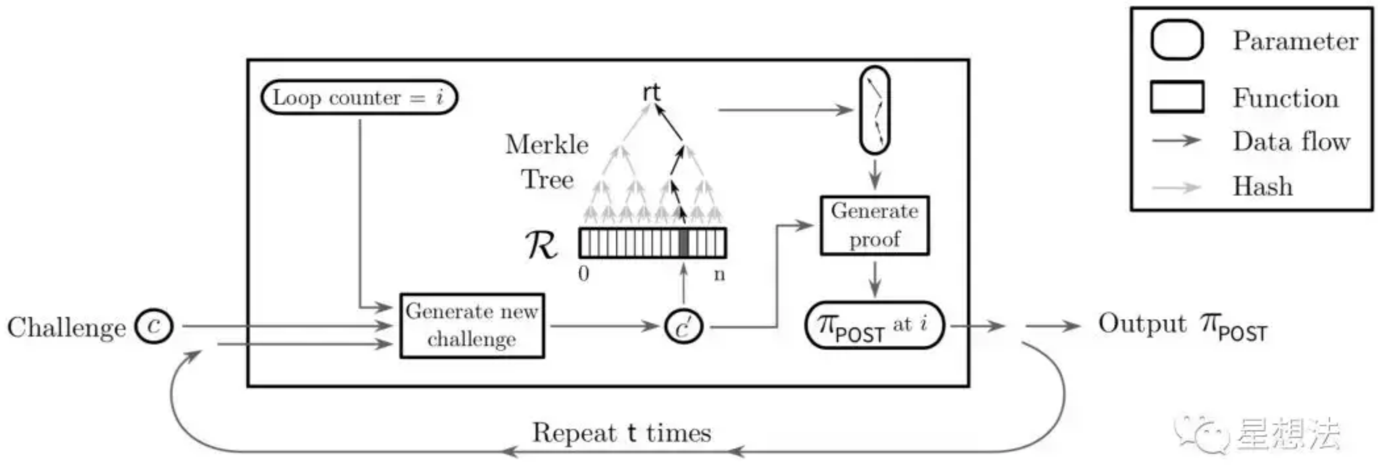


The VDFPostCompound::setup function sets the verification related parameters. The VDFPostCompound::prove function requires four parameters:

a. Parameters set by the setup function
b. Public data (comm_r data list, random challenge information)
c. Private data (Seal data structure information of Merkle tree)
d. Groth related parameter information

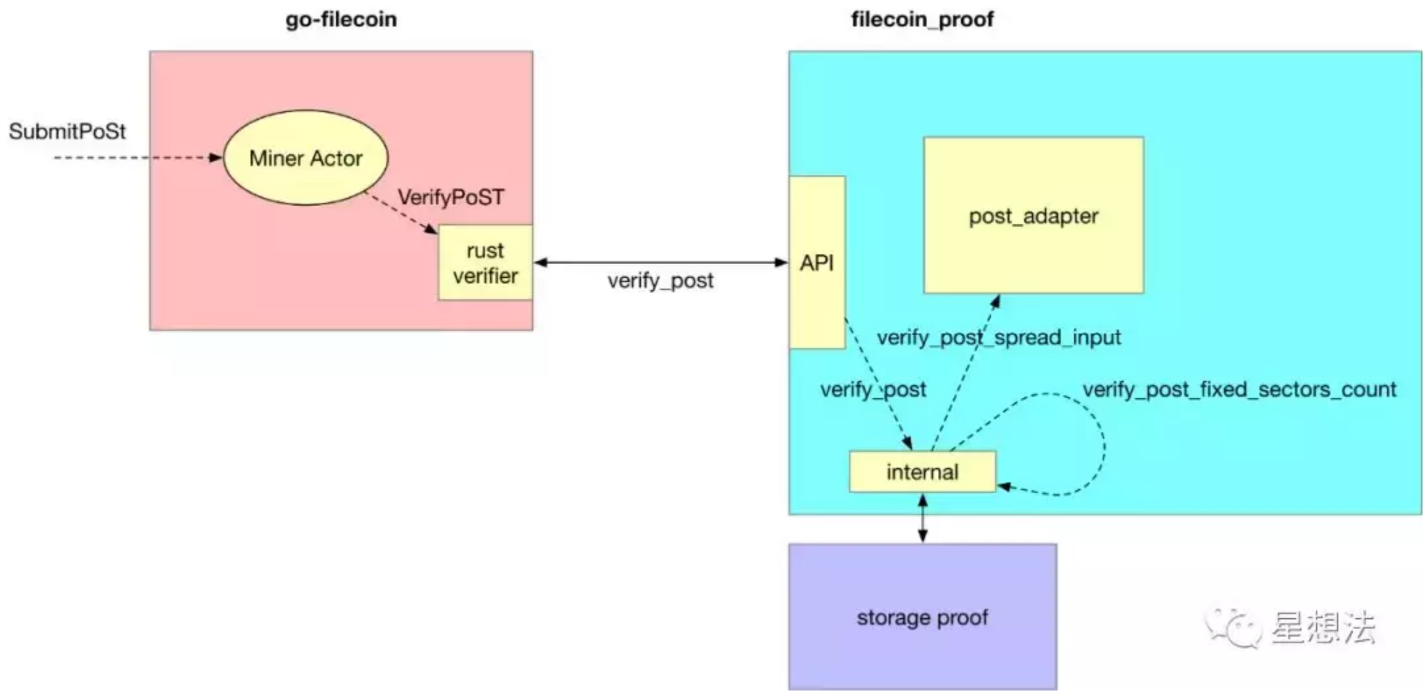
For the specific implementation of the source code, please view `rust-fil-proofs/storage-proofs/src/api/internal.rs` the `generate_post_fixed_sectors_count` function.

The logic implemented by the prove function is the framework diagram of PoSt in the white paper:



02 PoSt verification logic

After a storage miner generates a PoSt proof, it submits the proof information to the blockchain through the SubmitPoSt interface. When all miners of the Filecoin blockchain receive the SubmitPoSt Message transaction, they call VerifyPoST for copy verification. The general process is as follows:



For the specific implementation of the source code, please view `rust-fil-proofs/storage-proofs/src/api/internal.rs` the `verify_post_fixed_sectors_count` function.

Summary: PoRep is a proof of data storage in a certain Sector, once per Sector. PoSt is a storage certificate of a series of Sectors that have been sealed, once every once in a while. The core of the two proofs is the Groth16 zero-knowledge verification algorithm, based on the Bellman project.

Published on 2019-03-29 23:32 Reading (1055) Credits (5)
Category: FileCoin (https://learnblockchain.cn/categories/FileCoin)

Articles you may be interested in

Why is NFT different? How does Filecoin's distributed storage solution empower NFT? (https://learnblockchain.cn/article/2495) 21 views

IPFS Weekly 132 | The next gathering will showcase the cooperation between IPFS and NFT (https://learnblockchain.cn/article/2418) 69 views

IPFS helps expand ETH, Filecoin and DeFi to create the future together, and analyzes the powerful combination of IPFS and ETH (https://learnblockchain.cn/article/2390) 66 views

The Web3.0 China Summit came to a successful conclusion, Hu Feng, COO of Time Cloud: Filecoin needs long-termism! (https://learnblockchain.cn/article/2375) 94 views

Web 3.0 is coming, distributed storage plays an important role|Space Cloud invites you to participate in the Web3.0 China Summit and Distributed Storage Industry Conference (https://learnblockchain.cn/article/2349) 132 views

People's Daily Online: "Distributed storage opens up a market of 100 billion yuan", IPFS welcomes the new digital era! (https://learnblockchain.cn/article/2320) 186 views

Related questions

Which RPC interfaces are the three interface calls in the figure below the filecoin block explorer? ? ? (https://learnblockchain.cn/question/1505) 1 answer

What is the current progress of Filecoin? (https://learnblockchain.cn/question/4) 1 answer

0 comments