GEVULOT

HOW TO START PROVING ON GEVULOT?

GEVULOT.COM

WHY DECENTRALIZATION MATTERS?

Problem statement: the proving market is susceptible to centralization.

DESIGN CONSIDERATIONS

Workload allocation mechanisms:

- Auctions: prover undercutting is a reality
- Proof racing: resourceful entities could create monopoly
- Randomness: more neutral and fair allocation

Higher risk of centralization may mean:

- Increased liveness risk
- Higher censorship risk
- Losing permissionless and trustless properties

Decisions on where to outsource proof generation may come with trade-offs we need to be aware of.

CORE VALUES

Decentralization-focused design decisions ensure:

- No one can control where proving happens
- Liveness guarantees
- Censorship resistance
- Trustlessness
- Fair distribution of workloads and rewards
- Credible neutrality

At Gevulot, we believe in a ZK-future where proving is truly decentralized, performant, and cheap. Gevulot has been designed with this vision in mind.

FULL-STACK ZK CLOUD OFFERING

Gevulot is a decentralized compute network optimized for zero-knowledge proof generation and verification.

1 PROVING

Cheap and performant decentralized proving for any proof system.

- Consistent, low fees
- High availability
- Execution guarantees

2. VERIFICATION

Automatic verification and optional attestation to external networks.

- Lightweight & low cost native verification
- Additional security guarantees available through partners

3 CUSTOMISATION

Include any external software for custom functionality.

- State retrieval & broadcast
- TEEs, FPGAs & ASICs
- Non-ZK based verification

DECENTRALIZED ZK CLOUD

Gevulot is a decentralized proving layer that can legitimately compete with centralized options in both speed and cost.

1 CHEAP

Gevulot aggregates proving workloads from across the industry to better utilize the hardware resources in the network, thus improving the cost-structure for all participants.

2. PERFORMANT

Gevulot is exclusively optimized for the job of proving and verification. By removing all blockchain features which don't serve this purpose, we achieve centralized-equivalent performance for many types of workloads.

3 FLEXIBLE

Gevulot supports permissionless deployment of arbitrary provers. Provers can be written in a variety of languages, such as Rust or C++ and can utilize optimizations familiar from centralized setups, like multi-threading and GPU proving.

GEVULOT: OPTIMIZED FOR ZK-PROVING

Permissionless and programmable blockchain for ZK allowing the deployment of arbitrary proof-systems as on-chain programs.

PROVER-CENTRIC

- Aggregates workloads across different applications
- Maximizing hardware utilization and revenue for prover nodes

PROGRAMS

- Programs come in two varieties:
  - Provers
  - Verifiers

OFFERING

- Low cost
- High performance
- High liveness & availability
- Permissionless participation
- Credible neutrality

BLOCKS

- Include transactions and proofs
- No smart contract state and re-execution
- Verification of computation through validating ZKPs

NETWORK ACTORS

VALIDATORS

Main tasks:
- Processing transactions
- Broadcasting workloads to provers
- Consensus on block content and replicated state

Block building:
- Leader selected for every block
- Orders transactions and proofs into blocks

Incentives:
- Small transaction fee (to prevent spam)
- Block reward

PROVERS

Main tasks:
- Proof generation by the selected prover
- Verification of proofs by a subset of provers

Participation:
- Staking
- Capacity verification
  - Initial and periodic random PoW tasks

Workload distribution:
- Random selection
- Accept or decline
- Customizable redundancy
- Fallback mechanism

PROVING WORKLOADS

THE LIFECYCLE OF PROVING WORKLOADS

1. The user sends a *Run* transaction to the network with information necessary to execute a proving workload.
2. The receiving node adds the transaction to the mempool for inclusion in the next block.
3. Once the transaction has been included in a block and finalized, it is randomly allocated to a prover node.
4. The prover node completes the workload and sends the proof back to the mempool.
   a. The proof can already be shared with the user if verification and settlement are done on other settlement layers.
5. The proof gets included in a block, and becomes available for verification.
6. A random subset of prover nodes participate in the verification of the proof and vote on its validity.
7. Once all selected provers have verified the proof, the leader includes the verification in the next block.
8. Thus it reaches finality on Gevulot, and the reward is distributed to the prover.

PROVER ECONOMICS

PROVING WORKLOAD FEES

Two components:

- Small transaction fee
- Compute fee

The user specifies:

- Resource requirements for the prover workload
- Maximum compute time

USER FEE REBATE

Proportional to the elapsed time in the proof generation

PROVER REWARDS

Calculation:

- Prover Reward = Workload Fee - Fee Rebate

Provers are incentivized to improve proving speed and minimize rebates

FEE CALCULATION

- Fee = Tx Fee + (Compute Fee * Compute Time * Resource Requirement Multiplier - Fee Rebate)

CUSTOM PROVER SETS

USE CASES

Specialized hardware integration:

- Allows for seamless integration of custom hardware for intensive computational tasks

Interoperability via proxy nodes:

- Enhances interoperability with other blockchain platforms,
- Facilitating efficient
  - data management,
  - data storage and availability

Managing external software dependencies

PARTICIPATION

Voluntary participation:

- Provers can opt-in to support multiple custom prover sets
- Enhancing network resilience

Verification via PoW tasks:

- To ensure that provers meet hardware and software requirements

Transparency:

- Info available on active prover nodes within a custom prover set

GEVULOT

CUSTOM PROVER SETS

TECHNICAL ASPECTS

Efficient communication:
- Between the external program & the prover programs via a virtual, isolated network within a Gevulot node

Flexibility configuration:
- Using interfaces such as TUN/TAP, and
- Various protocols: gRPC, JSON-RPC etc.

Service discovery:
- Through mDNS or config details passed through the task definition

Maintaining security and integrity

PRICING AND INCENTIVES

Proof pricing:
- Determined by the prover program deployer
- Fair and flexible pricing
- Minimum cost to prevent undercutting

Network rewards:
- No subsidy by default
- Proportional to the percentage of the global prover set participating in the custom prover set

GEVULOT DEVNET - LIVE

The Gevulot devnet is a permissioned network for high-performance proof generation, and includes the full proof generation pipeline.

OVERVIEW

The devnet offers the following functionality:

1. Deploy arbitrary provers and verifiers
2. Run proving workloads to generate and verify proofs
3. Track workload status and retrieve outputs via an API
4. Store proofs

FEATURES

High performance nodes:

- Compute capacity to allow parallel proving or verifying.
- Ability to handle large input data (gigabytes).

Redundancy:

- Each node can function standalone.
- As long as even one devnet node is running, users will be able to generate proofs.

FREE for all registered users.

Devnet nodes are currently operated by Supranational, P2P.org, Staking Facilities & Rockaway X Infrastructure

DEVNET DASHBOARD

GEVULOT LIVE

1W 1M 6M 1Y

SEARCH

LIGHT DARK

1K REGISTERED USERS	+0.00%
22 PROVERS DEPLOYED	+0.00%
872K PROOFS GENERATED	+0.00%
3M PROOF VERIFICATIONS	+0.00%

STATE	TRANSACTION ID	PROVER ID	TIME
VERIFYING	CDB8B9C6525D6206033F9A89D357F8D694DCCADBC613E7EE69A9B9BB871D86DE	#PROVER 739468E524ED20A753BDE53BAD343AAC987412483F160DB8070F44185B61B557	10:09 AM, 22/05/24 →
PROVING	CDB8B9C6525D6206033F9A89D357F8D694DCCADBC613E7EE69A9B9BB871D86DE	#PROVER 739468E524ED20A753BDE53BAD343AAC987412483F160DB8070F44185B61B557	10:09 AM, 22/05/24 →
SUBMITTED	CDB8B9C6525D6206033F9A89D357F8D694DCCADBC613E7EE69A9B9BB871D86DE	#PROVER 739468E524ED20A753BDE53BAD343AAC987412483F160DB8070F44185B61B557	10:09 AM, 22/05/24 →
PROVING	A753E7DDF839BE573FA3C024EE7F5C77791D95A76FC2BB92A4CF0170176BDE76	#PROVER 739468E524ED20A753BDE53BAD343AAC987412483F160DB8070F44185B61B557	10:04 AM, 22/05/24 →
VERIFYING	D28CC6DFFAB042E602522A5261CD4A41D96191044A64057DA9CA2AE3C2F2CB82	#PROVER 739468E524ED20A753BDE53BAD343AAC987412483F160DB8070F44185B61B557	10:09 AM, 22/05/24 →
PROVING	D28CC6DFFAB042E602522A5261CD4A41D96191044A64057DA9CA2AE3C2F2CB82	#PROVER 739468E524ED20A753BDE53BAD343AAC987412483F160DB8070F44185B61B557	10:09 AM, 22/05/24 →
SUBMITTED	D28CC6DFFAB042E602522A5261CD4A41D96191044A64057DA9CA2AE3C2F2CB82	#PROVER 739468E524ED20A753BDE53BAD343AAC987412483F160DB8070F44185B61B557	10:09 AM, 22/05/24 →
VERIFYING	49BE658E80AB7F857AA3FEF1AD10A25243C30285B45609719B81EE14C7267787	#PROVER 739468E524ED20A753BDE53BAD343AAC987412483F160DB8070F44185B61B557	10:08 AM, 22/05/24 →
PROVING	49BE658E80AB7F857AA3FEF1AD10A25243C30285B45609719B81EE14C7267787	#PROVER 739468E524ED20A753BDE53BAD343AAC987412483F160DB8070F44185B61B557	10:08 AM, 22/05/24 →
SUBMITTED	49BE658E80AB7F857AA3FEF1AD10A25243C30285B45609719B81EE14C7267787	#PROVER 739468E524ED20A753BDE53BAD343AAC987412483F160DB8070F44185B61B557	10:08 AM, 22/05/24 →
VERIFYING	8C3D4498C1CE3922CD0CEC94D758793F86096AAB509DB627307492F73A57FF0B	#PROVER 739468E524ED20A753BDE53BAD343AAC987412483F160DB8070F44185B61B557	10:08 AM, 22/05/24 →
PROVING	8C3D4498C1CE3922CD0CEC94D758793F86096AAB509DB627307492F73A57FF0B	#PROVER 739468E524ED20A753BDE53BAD343AAC987412483F160DB8070F44185B61B557	10:08 AM, 22/05/24 →
SUBMITTED	8C3D4498C1CE3922CD0CEC94D758793F86096AAB509DB627307492F73A57FF0B	#PROVER 739468E524ED20A753BDE53BAD343AAC987412483F160DB8070F44185B61B557	10:08 AM, 22/05/24 →
VERIFYING	FF232CC454D993AE048CFDCAADA6F5E2E8BD20632FE8AF2D96BFAC5A41979489	#PROVER 739468E524ED20A753BDE53BAD343AAC987412483F160DB8070F44185B61B557	10:08 AM, 22/05/24 →

Twitter Docs Telegram Substack Github We're hiring

KEY REGISTRATION

The Gevulot Devnet is permissioned. All users need to register a key for whitelisting to get access.

REQUIREMENTS

Prerequisites to be installed:

- Rust
- Cargo
- C Development Tools (GCC, Make, Binutils etc.)
- openssl-devel (libssl-dev in Debian derivatives)
- protobuf-compiler

KEY GENERATION & REGISTRATION

1. **Install the Gevulot CLI:**
   - cargo install --git https://github.com/gevulotnetwork/gevulot.git gevulot-cli

   (Note: It should be in **$PATH**, and can be issued by calling gevulot-cli.)
2. **Generate a new cryptographic key pair** which will be used to sign transactions.
   - gevulot-cli generate-key
3. A local file **localkey.pki** is created in the current directory and the public key is printed for copying.

Key registration: submit your key through the registration form.

GEVULOT

PROVER/VERIFIER INTEGRATION

RUNNING ENVIRONMENT

Nanos Unikernel:

- Provides a very lightweight operating system.

INTEGRATION PROCESS

Requirements:

- Programs should be
  - compatible with Linux x86_64 architecture, and
  - communicate via gRPC protocol.

Shim Library:

- Helper library in Rust, with C bindings.
- Provides functions to facilitate simple integration.

RUST IMPLEMENTATION

Modifications required:

- **Callback function** for running the tasks.
- **Shim delegation:** A new **main()** function to delegate control to the **shim**.

SAMPLE CODE

Full details with example and code snippets:

- Gevulot Docs - DEVNET section:
  - https://docs.gevulot.com/gevulot-docs/devnet/integration

PROVER/VERIFIER PACKAGING

CONFIGURATION

Environment:

- Each program runs in a VM under Nanos Unikernel.

Manifest file:

- Contains configuration data on how the system should run the program.

Configuration example:

- https://docs.gevulot.com/gevulot-docs/devnet/prover-verifier-packaging#manifest

PACKAGING

Packaging steps:

- **Install Ops**: Tool for working with Nanos.
- **Compile program**: Must target Linux x86_64 architecture.
- **Build** program **image**: Use Ops to create a bootable image.

Complete example and code snippets:

- Gevulot Docs - DEVNET section:
  - https://docs.gevulot.com/gevulot-docs/devnet/prover-verifier-packaging#complete-example

PROVER/VERIFIER DEPLOYMENT

DEPLOYMENT

Prerequisites:

- **Install Gevulot CLI:** Required for deployment commands.
- **Register key:** Ensure your key is registered and whitelisted.

Deployment steps:

- **Calculate hash for programs:** For integrity verification.
- **Upload programs:** Host files on an HTTP server.
- **Deploy programs:** Use the CLI to deploy both prover and verifier.

USEFUL INFO

Deployment:

- One deployment always consists of prover & verifier

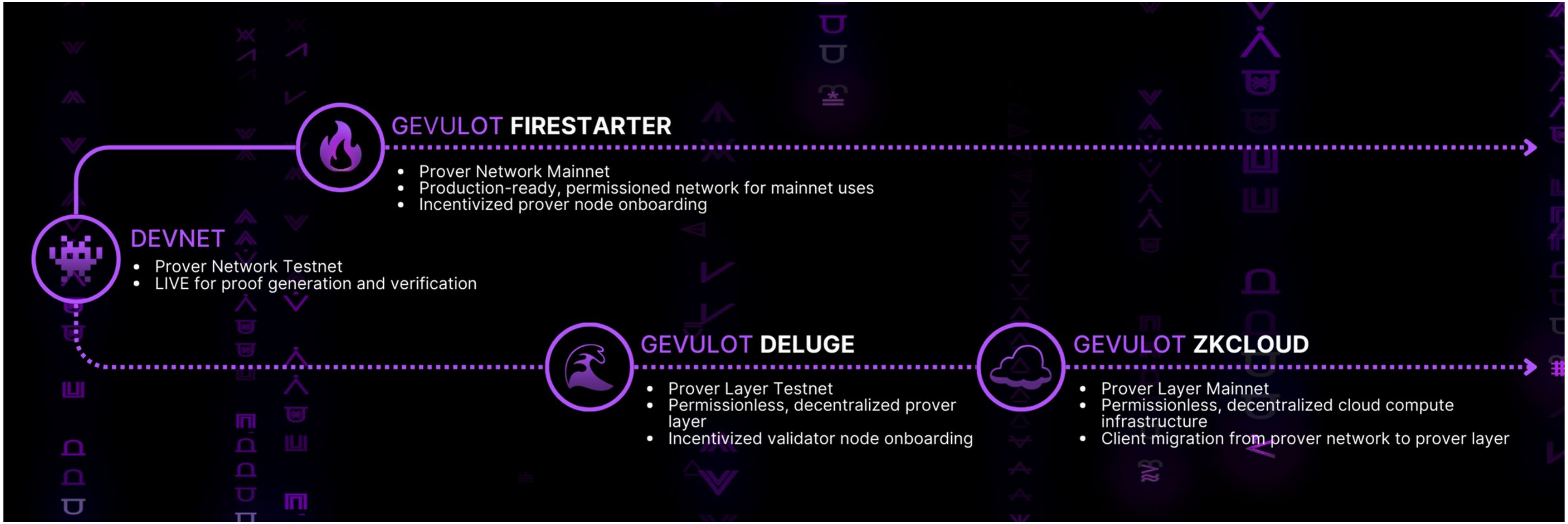
Resource requirements:

- Default resource allocation
- Customizable resource allocation: deployer can specify resource requirements for the program.

Example and commands:

- Gevulot Docs - DEVNET section:
    - https://docs.gevulot.com/gevulot-docs/devnet/deployment

GEVULOT PRODUCT ROADMAP



JOIN OUR DEVNET AT GEVULOT.COM

GEVULOT