Evice Blockchain: Arsitektur Blockchain Hibrida Layer 1 & 2 yang Skalabel, Aman, dan Siap Masa Depan

Syafiq Nabil Assirhindi

20 Oktober 2025

Ringkasan

Evice Blockchain adalah platform blockchain Layer 1 (L1) berkinerja tinggi yang dirancang untuk mengatasi trilema blockchain—skalabilitas, keamanan, dan desentralisasi. Dengan mengintegrasikan solusi penskalaan Layer 2 (L2) Zero-Knowledge Rollup (ZK-Rollup) secara native, Evice menawarkan throughput transaksi yang masif dengan tetap mewarisi keamanan dari lapisan dasarnya. Arsitektur ini didukung oleh protokol konsensus Proof-of-Stake (PoS) hibrida inovatif bernama *Aegis*, yang menggabungkan konfirmasi transaksi cepat melalui lapisan *Velocity* dengan finalitas absolut yang dijamin oleh lapisan *Gravity*. Platform ini mendukung smart contract melalui runtime WebAssembly (WASM) yang aman dan efisien [16], serta mengadopsi skema kriptografi canggih, termasuk tanda tangan *post-quantum* Dilithium [8] untuk ketahanan jangka panjang. Whitepaper ini menguraikan arsitektur teknis, mekanisme konsensus, solusi L2, dan komponen ekosistem yang menjadikan Evice sebagai fondasi yang kuat untuk aplikasi terdesentralisasi generasi berikutnya.

1 Pendahuluan

Dalam beberapa tahun terakhir, teknologi blockchain telah menunjukkan potensi revolusioner di berbagai industri. Namun, adopsi massal masih terhambat oleh tantangan fundamental yang dikenal sebagai trilema blockchain: sulitnya mencapai skalabilitas tinggi, keamanan yang kuat, dan desentralisasi penuh secara bersamaan. Banyak platform yang ada mengorbankan salah satu aspek untuk mengoptimalkan yang lain, yang membatasi kasus penggunaannya. Akibatnya, pengguna mengalami biaya tinggi, waktu tunggu lama, dan risiko keamanan.

Evice Blockchain dirancang dari awal untuk mengatasi tantangan ini secara holistik. Visi kami adalah menciptakan platform yang tidak hanya cepat dan murah untuk pengguna akhir, tetapi juga sangat aman, terdesentralisasi, dan siap menghadapi tantangan komputasi masa depan, termasuk ancaman dari komputasi kuantum.

Untuk mencapai ini, Evice mengimplementasikan arsitektur hibrida:

- **Layer 1 (Lapisan Dasar):** Berfungsi sebagai lapisan penyelesaian (*settlement layer*) dan sumber kebenaran (*source of truth*). Lapisan ini bertanggung jawab atas keamanan jaringan melalui konsensus PoS, finalitas transaksi, dan eksekusi *smart contract* penting.
- **Layer 2 (Lapisan Penskalaan):** Beroperasi sebagai ZK-Rollup yang mengeksekusi ribuan transaksi di luar rantai (*off-chain*), menghasilkan bukti kriptografis (ZK-SNARK) atas validitasnya, dan mengirimkan bukti tersebut ke Layer 1. Hal ini secara dramatis meningkatkan throughput dan mengurangi biaya transaksi.

Whitepaper ini akan mengupas tuntas setiap komponen teknis yang membangun Evice Blockchain, mulai dari arsitektur dasarnya hingga detail implementasi protokol konsensus dan solusi Layer 2.

2 Arsitektur Keseluruhan

Arsitektur Evice dirancang secara modular, memisahkan lapisan eksekusi dari lapisan konsensus dan penyelesaian. Alur interaksi antar komponen utama digambarkan di bawah ini, diikuti oleh penjelasan rinci untuk setiap peran.

- **Pengguna (User):** Aktor eksternal yang berinteraksi dengan jaringan. Pengguna mengirimkan transaksi L1 ke endpoint RPC gRPC milik Node L1, atau mengirimkan transaksi L2 ke endpoint JSON-RPC milik Node L2 (Sequencer).
- **Node L1 (Validator):** Partisipan jaringan yang menjalankan perangkat lunak `evice_blockchain`. Mereka bertanggung jawab untuk memvalidasi transaksi, berpartisipasi dalam konsensus Aegis, mengusulkan blok baru, dan menjaga integritas state L1. Node ini juga memvalidasi dan mengeksekusi transaksi L2 Rollup yang dikirim oleh Sequencer.
- **Node L2 (Sequencer):** Entitas khusus yang dipilih di L1 melalui mekanisme *Stake-Weighted VRF*. Sequencer menjalankan binary `sequencer` yang mengekspos API JSON-RPC untuk menerima transaksi L2. Tugasnya adalah mengurutkan transaksi (berdasarkan *priority fee*), membuat *batch*, memanggil *Prover*, mengumpulkan tanda tangan dari DAC, dan akhirnya membungkus semua data ini ke dalam sebuah transaksi L1 jenis `SubmitRollupBatch`.
- **Prover & Aggregator:** *Prover* adalah komponen yang mengambil *batch* transaksi dan menghasilkan bukti kriptografis ZK-SNARK yang membuktikan validitas transisi state L2. *Aggregator* kemudian dapat menggabungkan beberapa bukti dari *batch* yang berurutan menjadi satu bukti agregat, yang secara signifikan mengurangi biaya verifikasi di L1.

- **Data Availability Committee (DAC):** Sekelompok entitas tepercaya yang bertanggung jawab untuk menerima data *batch* dari Sequencer dan menandatanganinya. Kumpulan tanda tangan dari DAC disertakan dalam transaksi `SubmitRollupBatch` ke L1 sebagai jaminan bahwa data transaksi L2 tersedia dan dapat dipulihkan jika Sequencer menjadi tidak responsif.

Interaksi antara L1 dan L2 difasilitasi oleh dua mekanisme utama. Pertama, melalui transaksi `SubmitRollupBatch` atau `SubmitAggregateRollupBatch` yang membawa pembaruan state L2 ke L1. Kedua, melalui *smart contract bridge* di L1, yang mengelola deposit aset ke L2 dan memproses penarikan dari L2 dengan memverifikasi `WithdrawalProof` (bukti Merkle inklusi) terhadap *state root* L2 yang telah dicatat di L1.

3 Layer 1: Evice Blockchain Core

Lapisan dasar Evice adalah fondasi keamanan dan desentralisasi seluruh ekosistem.

3.1 State Machine

Manajemen state di Evice tidak hanya menggunakan konsep Merkle Patricia Trie [18], tetapi secara spesifik diimplementasikan menggunakan pustaka `trie-db` dengan layout kustom `EviceTrieLayout`. Layout ini mendefinisikan `Keccak-256` [7] sebagai fungsi *hashing* untuk node trie dan menggunakan `ProductionNodeCodec` untuk serialisasi node yang efisien. Sebagai backend penyimpanan persisten, Evice memanfaatkan `ParityDB` (sebuah implementasi key-value store berbasis `RocksDB` yang dioptimalkan [2, 11]). Akses ke database diabstraksi melalui `ParityDbTrieBackend`, yang mengelola penulisan *pending* dan *caching* node dalam memori menggunakan `LruCache` untuk mempercepat akses state. Modifikasi state melalui `TrieSession` memastikan atomisitas; perubahan hanya di-commit ke `ParityDB` setelah blok berhasil dieksekusi.

3.1.1 Eksekusi Spekulatif dan State Sementara

Selain `BlockTree` yang mengelola struktur cabang, Evice juga memanfaatkan `SpeculativeChain`. Komponen ini bertanggung jawab untuk mengeksekusi blok-blok yang diterima secara optimis, bahkan sebelum mereka tentu menjadi bagian dari rantai kanonis atau difinalisasi. Ketika sebuah blok baru ditambahkan ke `SpeculativeChain` (setelah validasi awal), transisinya dieksekusi terhadap state dari blok induknya (yang mungkin juga spekulatif). Hasil eksekusi (*overlay* perubahan state atau *post-state root*) disimpan dalam *cache* (`LruCache`).

Tujuannya adalah untuk menyediakan akses cepat ke state terbaru yang *kemungkinan* akan menjadi kanonis, misalnya untuk merespons query RPC yang meminta informasi akun pada blok terbaru yang belum final. `SpeculativeChain` menggunakan `TrieSession` untuk mengelola state sementara ini secara efisien

tanpa langsung menulis ke database utama. Ketika sebuah blok difinalisasi oleh lapisan Gravity, `SpeculativeChain` akan memperbarui *finalized head*-nya dan dapat memangkas cabang-cabang spekulatif yang tidak relevan lagi.

3.2 Runtime Smart Contract WASM

Evice mendukung *smart contract* yang dikompilasi ke WebAssembly (WASM) [16], sebuah standar biner yang portabel, efisien, dan aman.

- **Lingkungan Eksekusi:** Kontrak dijalankan dalam *sandbox* menggunakan *runtime* `Wasmer` [17], yang memberikan isolasi kuat antara eksekusi kontrak dan sistem host.
- **Metering (Gas):** Setiap instruksi WASM dan pemanggilan fungsi host memiliki biaya gas yang telah ditentukan. Eksekusi akan berhenti jika gas yang disediakan habis, mencegah *loop* tak terbatas dan serangan DoS.
- **API Kontrak (Fungsi Host):** Kontrak dapat berinteraksi dengan state blockchain melalui serangkaian API aman yang diekspos oleh *runtime*, termasuk:
  - `read_storage` & `write_storage`: Membaca dan menulis ke penyimpanan persisten kontrak.
  - `get_caller`: Mendapatkan alamat yang memanggil kontrak.
  - `get_block_timestamp`: Mendapatkan timestamp blok saat ini.
  - `revert`: Menghentikan eksekusi dan mengembalikan perubahan state.
  - `log`: Menerbitkan *event* yang dapat diindeks oleh layanan eksternal.
  - `transfer_native_token`: Meminta transfer token native dari saldo kontrak.

3.3 Serialisasi Data Kontrak

Runtime WASM Evice memberikan fleksibilitas dalam format serialisasi data yang digunakan oleh smart contract, baik untuk *call data* (input fungsi) maupun untuk data yang disimpan dalam *storage*. SDK Kontrak (`evice-contract-sdk`) secara default menggunakan `borsh` [9] untuk serialisasi state yang disimpan karena efisiensi dan kompatibilitasnya dengan lingkungan `no_std`. Namun, untuk *call data*, kontrak dapat memilih format lain yang lebih sesuai dengan ekosistem *off-chain* atau *tooling* yang ada. Sebagai contoh, implementasi kontrak ERC-721 menggunakan `serde_json` [14] untuk mendefinisikan antarmuka pemanggilannya, sementara kontrak Fungible Token dan Bridge menggunakan `borsh`. Pengembang disarankan untuk memilih format serialisasi *call data* yang paling sesuai untuk kasus penggunaan mereka, sementara penggunaan `borsh` direkomendasikan untuk data state internal kontrak demi efisiensi *on-chain*.

3.4 Mekanisme Fee

Untuk mengatur penggunaan sumber daya jaringan dan memberikan insentif kepada validator, Evice mengimplementasikan mekanisme fee yang mirip dengan EIP-1559 [3]:

- `base_fee_per_gas`: Biaya dasar per unit gas yang disesuaikan secara algoritmis di setiap blok berdasarkan kepadatan blok sebelumnya. *Base fee* ini "dibakar" (dihapus dari peredaran).
- `max_priority_fee_per_gas`: "Tip" yang diberikan oleh pengguna kepada proposer blok sebagai insentif untuk memprioritaskan transaksi mereka.

4 Protokol Konsensus: Aegis

Aegis adalah protokol konsensus hibrida yang dirancang untuk memberikan kecepatan dan finalitas yang terinspirasi oleh HotStuff [19]). Terdiri dari dua lapisan yang bekerja secara sinergis:

4.1 Lapisan Velocity (Konfirmasi Cepat)

Pemilihan *sub-committee* untuk setiap ronde konsensus dilakukan oleh fungsi `determine_sub_committee`. Proses ini dimulai dengan mengambil daftar validator aktif saat ini dari state. Daftar ini kemudian diurutkan secara kanonis (misalnya, berdasarkan alamat) untuk memastikan determinisme. Sebuah *seed* acak-semu dihasilkan dengan menggabungkan *hash* dari `highest_seen_qc` (Quorum Certificate tertinggi yang disepakati dari ronde sebelumnya) dengan nomor ronde saat ini, lalu di-*hash* menggunakan SHA-256. *Seed* ini digunakan untuk menginisialisasi *Random Number Generator* (RNG) yang kemudian digunakan untuk mengacak (*shuffle*) urutan daftar validator yang sudah diurutkan. Sejumlah `AEGIS_SUB_COMMITTEE_SIZE` (nilai dari parameter genesis validator pertama dari daftar yang telah diacak inilah yang membentuk *sub-committee* untuk ronde tersebut. Proposer kemudian berotasi di antara anggota *sub-committee* ini berdasarkan nomor *step* dalam ronde. Jika proposer pada *step* n gagal mengusulkan blok dalam `PROPOSER_TIMEOUT` (sekitar 1200ms), konsensus akan maju ke *step* $n + 1$ dan validator berikutnya dalam urutan *sub-committee* menjadi proposer.

4.1.1 Manajemen Fork dengan BlockTree

Untuk menangani kemungkinan adanya *fork* atau cabang sementara selama lapisan Velocity sebelum finalitas dicapai oleh Gravity, Evice menggunakan struktur data `BlockTree`. Struktur ini menyimpan semua blok valid yang diterima node, bahkan jika blok tersebut bukan bagian dari rantai terpanjang saat ini. Setiap blok direpresentasikan sebagai `BlockNode`, yang menyimpan data blok itu sendiri, *hash* induknya, status pemrosesan state (`BlockNodeStatus`), dan *post-state root* (jika state sudah berhasil dieksekusi).

`BlockTree` melacak dependensi antar blok; sebuah blok tidak dapat diproses statenya hingga state induknya berstatus `StateReady`. Struktur ini juga menghitung *weight* (bobot) untuk setiap cabang berdasarkan jumlah suara konsensus (`VelocityVote`) yang diterima oleh blok-blok di cabang tersebut. Fungsi `find_head` digunakan untuk menentukan ujung rantai (*head*) kanonis berdasarkan cabang dengan bobot tertinggi yang state-nya sudah siap. Ketika lapisan Gravity memfinalisasi sebuah blok, `BlockTree` akan melakukan *pruning* dengan menghapus semua node yang tidak lagi merupakan leluhur atau turunan dari blok yang difinalisasi, memastikan efisiensi memori.

4.1.2 Pemrosesan State Asinkron

Untuk meningkatkan throughput dan responsivitas node, eksekusi state yang intensif komputasi (terutama validasi transaksi dan eksekusi smart contract WASM) dilakukan secara asinkron. Ketika `BlockTree` menerima blok baru yang induknya sudah `StateReady`, ia tidak langsung memproses state blok baru tersebut di *thread* utama konsensus. Sebaliknya, ia men-spawn sebuah *background task* baru (menggunakan `tokio::spawn` atau `tokio::task::spawn_blocking`) [15].

Task ini bertanggung jawab untuk memanggil fungsi `apply_transactions_to_session` yang melakukan validasi transaksi mendalam, eksekusi WASM, dan perhitungan *state root* baru dalam `TrieSession` terpisah. Hasilnya (sukses dengan *post-state root* baru, atau gagal dengan error) kemudian dikirim kembali ke *thread* utama melalui *channel* `mpsc` sebagai `BlockProcessingResult`. *Thread* utama kemudian memperbarui status `BlockNode` yang sesuai di `BlockTree`. Pendekatan asinkron ini memungkinkan mesin konsensus untuk terus menerima dan memvalidasi proposal atau suara baru sementara state blok sebelumnya masih diproses di latar belakang.

4.2 Lapisan Gravity (Finalitas Absolut)

Lapisan ini memberikan finalitas deterministik (tidak dapat diubah) secara periodik.

- **Epoch dan Checkpoint:** Waktu dibagi menjadi *epoch* (misalnya, setiap 10 blok). Blok terakhir di setiap *epoch* dianggap sebagai *checkpoint*.
- **Manajemen Kunci BLS:** Setiap validator aktif diwajibkan memiliki pasangan kunci BLS (Boneh–Lynn–Shacham) [1] individual. Kunci-kunci ini umumnya dihasilkan secara offline oleh validator sebelum bergabung ke jaringan (misalnya, menggunakan perkakas bantu seperti validator-tool). Kunci publik BLS milik validator kemudian didaftarkan dan disimpan dalam state on-chain yang terkait dengan akun validator tersebut, sedangkan kunci privat BLS disimpan secara aman oleh masing-masing validator dan digunakan untuk operasional node. Mekanisme DKG (Distributed Key Generation) secara dinamis belum diaktifkan dalam implementasi saat ini.

- **Voting Finalitas:** Setelah sebuah *checkpoint* diusulkan dan diproses, semua validator aktif menandatangani hash dari *checkpoint* tersebut menggunakan kunci privat BLS individual mereka. Tanda tangan parsial ini disebut sebagai `FinalityVote`.

- **Finality Certificate:** Kumpulan `FinalityVote` (tanda tangan BLS individual) dari para validator dikumpulkan. Apabila jumlah suara yang valid dari validator unik telah mencapai ambang batas kuorum (lebih dari 2/3 validator aktif), tanda tangan-tanda tangan individual tersebut diagregasi secara kriptografis menjadi satu tanda tangan BLS agregat yang ringkas. Tanda tangan agregat ini, beserta hash *checkpoint* dan daftar validator pemberi suara, membentuk `FinalityCertificate`. Sertifikat ini menyajikan bukti matematis yang efisien dan tidak dapat disangkal bahwa mayoritas validator telah menyetujui *checkpoint*, sehingga memberikan finalitas absolut pada *checkpoint* tersebut dan seluruh blok dalam sejarahnya. Verifikasi sertifikat dapat dilakukan oleh node manapun dengan menggunakan kunci publik BLS individual para validator yang tersimpan dalam state.

4.3 Mekanisme Slashing

Untuk menjaga keamanan jaringan, validator yang berperilaku jahat atau lalai akan dihukum melalui pemotongan stake (*slashing*).

- **Double Signing:** Jika seorang validator menandatangani dua blok berbeda pada ketinggian dan ronde yang sama, siapa pun dapat mengirimkan bukti melalui transaksi `ReportDoubleSigning`. Pelanggaran ini mengakibatkan pemotongan stake sebesar 10%.

- **Invalid State Transition:** Jika seorang validator mengusulkan blok dengan transisi state yang salah (misalnya, *state root* yang tidak valid), bukti dapat diajukan untuk menghukum mereka. Pelanggaran ini mengakibatkan pemotongan stake sebesar 50%.

- **Inactivity:** Validator yang sering offline dan gagal berpartisipasi dalam konsensus selama periode waktu tertentu (`INACTIVITY_THRESHOLD_BLOCKS`, saat ini 1000 blok). akan dikenai penalti ringan sebesar 1% dan akhirnya dikeluarkan dari set validator aktif (*jailed*).

5 Layer 2: ZK-Rollup

Untuk mencapai skalabilitas ekstrem, Evice mengintegrasikan ZK-Rollup sebagai solusi Layer 2.

5.1 Arsitektur

- **Sequencer:** Entitas yang dipilih di L1 untuk setiap slot *batch* L2 menggunakan mekanisme `Stake-Weighted VRF`. Algoritma pemilihan (`Stake-WeightedVrfSelector`) menghitung nilai acak semu untuk setiap sequencer kandidat (validator yang terdaftar sebagai sequencer) berdasarkan selection material (hash blok L1 sebelumnya) dan alamat kandidat. Nilai ini kemudian dibagi dengan jumlah stake kandidat. Kandidat dengan nilai terendah terpilih. Sequencer terpilih menjalankan node terpisah (sequencer.rs) yang menerima transaksi L2 melalui API JSON-RPC, mengurutkannya (berdasarkan priority fee), membuat *batch*, menghasilkan bukti VRF kepemimpinannya, memanggil `subprocess prover` untuk menghasilkan bukti ZK, mengumpulkan tanda tangan dari *Data Availability Committee* (DAC), dan mengirimkan transaksi `SubmitRollupBatch` ke L1.

- **Prover & Aggregator:** Setelah *batch* dibuat, Prover menghasilkan bukti ZK-SNARK untuk *batch* tersebut. Aggregator dapat mengambil beberapa bukti dari *batch* yang berurutan dan menggabungkannya menjadi satu bukti tunggal, yang jauh lebih murah untuk diverifikasi di L1.

5.2 Sirkuit Zero-Knowledge

- **Sirkuit Batch (Groth16/BLS12-377):** Sirkuit utama (`BatchSystem-Circuit`) memodelkan transisi state dari sebuah Merkle Tree berbasis hash Poseidon [5]. Sirkuit ini memvalidasi bahwa setiap transaksi dalam *batch* adalah valid (misalnya, pengirim memiliki saldo yang cukup) dan bahwa *final state root* adalah hasil yang benar dari penerapan semua transaksi pada *initial state root*.

- **Sirkuit Agregasi (Groth16/BW6-761):** Untuk efisiensi lebih lanjut, `AggregationCircuit` (l2_aggregation.rs) memverifikasi dua bukti L2 batch (l2_circuit.rs) di dalam sirkuit ZK lain (`Groth16/BW6-761`). Sirkuit ini memastikan kedua bukti valid dan *state root* akhir dari bukti pertama cocok dengan *state root* awal dari bukti kedua. Outputnya adalah satu bukti agregat yang lebih ringkas. Implementasi saat ini (aggregator.rs, sequencer.rs) mendukung agregasi satu tingkat (dua bukti menjadi satu).

5.3 Data Availability (DA)

Untuk memastikan bahwa siapa pun dapat merekonstruksi state L2 jika Sequencer berhenti bekerja, data transaksi harus tersedia. Evice menggunakan model *Data Availability Committee (DAC)* di mana sekelompok pihak tepercaya menandatangani *hash* data *batch*, membuktikan bahwa mereka telah menerima dan akan menyimpan data tersebut. Tanda tangan ini disertakan dalam transaksi L1.

5.4 Mekanisme Fee Layer 2

Meskipun whitepaper ini fokus pada mekanisme fee L1 yang mirip EIP-1559, lapisan L2 juga memiliki mekanisme fee internal yang dikelola oleh Sequencer. Pengguna yang mengirimkan transaksi L2 menentukan `max_fee_per_gas` dan `max_priority_fee_per_gas`. Sequencer menggunakan `max_priority_fee_per_gas` (tip prioritas) sebagai kriteria utama untuk mengurutkan transaksi yang akan dimasukkan ke dalam batch berikutnya. Transaksi dengan tip lebih tinggi akan diprioritaskan.

6 Kriptografi

Evice menggunakan tumpukan kriptografi modern dan beragam untuk mengamankan berbagai aspek protokol:

- **Tanda Tangan (L1):** *Dilithium* [8], salah satu algoritma yang distandarisasi oleh NIST untuk kriptografi *post-quantum*, digunakan untuk tanda tangan transaksi dan blok. Ini memberikan ketahanan terhadap serangan dari komputer kuantum di masa depan.
- **Agregasi Tanda Tangan (Finalitas):** *BLS (Boneh–Lynn–Shacham)* [1] digunakan untuk `FinalityCertificate` karena kemampuannya mengagregasi banyak tanda tangan menjadi satu tanda tangan tunggal yang ringkas.
- **Keacakan (Pemilihan Sequencer):** *Schnorrkel VRF (Verifiable Random Function)* [4] digunakan untuk pemilihan pemimpin sequencer yang tidak dapat diprediksi namun dapat diverifikasi.
- **Hashing:**
  - Keccak-256 [7]: Untuk alamat dan struktur Merkle Patricia Trie L1.
  - SHA-256: Untuk *hashing* blok, transaksi, dan data internal.
  - Poseidon [5]: Fungsi hash yang dioptimalkan untuk ZK-SNARK, digunakan dalam Merkle Tree L2 dan sirkuit ZK.
- **Zero-Knowledge Proofs:** *Groth16* [6], skema ZK-SNARK yang efisien, *dipilih karena menghasilkan bukti yang sangat ringkas dan verifikasi on-chain (L1) yang cepat.* Skema ini digunakan untuk bukti L2 dan agregasi[cite: 117].
- **Enkripsi:** *Scrypt* (KDF) [12] dan *XChaCha20Poly1305* (AEAD) [10] digunakan untuk mengamankan file *keystore*[cite: 118]. *Scrypt berfungsi sebagai KDF yang intensif memori untuk melindungi kata sandi dari serangan brute-force, sementara XChaCha20Poly1305 menyediakan enkripsi terotentikasi atas data kunci privat itu sendiri.*

7 Jaringan P2P

Komunikasi antar node diatur oleh `libp2p` [13], sebuah *framework* jaringan modular yang menyediakan:

- **Discovery:** Kademlia DHT dan protokol Identify untuk menemukan dan mengidentifikasi *peer* lain.
- **Transport:** TCP dan QUIC.
- **Propagasi Pesan:**
  - Gossipsub: Untuk menyebarkan transaksi dan pesan konsensus secara efisien ke seluruh jaringan.
  - Request-Response: Untuk permintaan data spesifik seperti blok, header, atau sinkronisasi state.
- **Manajemen Reputasi:** Node melacak perilaku *peer* dan akan memberikan penalti atau memutus koneksi *peer* yang berperilaku buruk.

Selain itu, node menggunakan `AddressBook` yang dipelihara secara lokal untuk memetakan alamat validator on-chain ke identitas jaringan (`PeerId` dan `Multiaddr`) mereka. Informasi ini diperbarui dari state chain terbaru dan digunakan untuk komunikasi langsung, misalnya, mengirim vote ke proposer atau pesan DKG. Node juga dapat bertukar daftar *peer* yang diketahui melalui protokol `Request-Response` untuk mempercepat penemuan.

8 Governance

Evice memiliki kerangka kerja *governance on-chain* yang memungkinkan pemegang token (melalui *stake*) untuk mengusulkan dan memberikan suara pada perubahan protokol.

- **Proposal:** Validator dapat mengajukan proposal menggunakan transaksi `SubmitProposal`. Struktur proposal saat ini mendukung beberapa jenis tindakan (`ProposalAction`), termasuk proposal teks, perubahan parameter jaringan (`UpdateParameter`), dan usulan *upgrade runtime* (`UpgradeRuntime`).
- **Voting:** Suara dihitung berdasarkan jumlah *stake* yang dimiliki oleh validator yang memberikan suara melalui transaksi `CastVote`. Setelah periode voting berakhir, protokol akan secara otomatis menghitung hasilnya. Proposal yang mencapai kuorum dan mayoritas "ya" akan ditandai sebagai disetujui, meskipun logika untuk eksekusi otomatis dari tindakan proposal tersebut (seperti mengubah parameter state) merupakan bagian dari pengembangan di masa depan.

9 Tokenomics

Ekonomi token Evice dirancang untuk menyelaraskan insentif semua peserta jaringan.

- **Staking:** Validator harus men-stake token untuk berpartisipasi dalam konsensus, mengamankan jaringan, dan mendapatkan imbalan.
- **Rewards:** Imbalan blok didistribusikan kepada proposer blok. Imbalan ini bersifat dinamis, disesuaikan berdasarkan total jumlah token yang di-stake di jaringan. Proposer juga menerima *priority fees* dari transaksi yang mereka sertakan.
- **Fee Burning:** *Base fee* dari setiap transaksi dibakar, menciptakan tekanan deflasi pada pasokan token seiring dengan meningkatnya penggunaan jaringan.
- **Slashing:** Validator yang melanggar aturan akan kehilangan sebagian dari stake mereka. Dana yang disita sepenuhnya akan dibakar.

10 Tools dan Ekosistem

Untuk mendukung pengembangan dan operasi jaringan, Evice menyediakan serangkaian *tool* yang komprehensif, termasuk:

- **Klien Node:** Implementasi node penuh `evice_blockchain` yang menjalankan semua komponen L1, termasuk P2P, konsensus, dan RPC.
- **Layanan Inti L2:**
  - **Sequencer:** Binary `sequencer` yang menjalankan node L2, menerima transaksi melalui API JSON-RPC, membuat *batch*, dan berkoordinasi dengan *prover*.
- **Utilitas CLI Pengguna & Validator:**
  - `create_tx`: Alat utama untuk membuat dan mengirim berbagai jenis transaksi L1 dan L2.
  - `create_keystore`: Untuk membuat file *keystore* baru yang dienkripsi dengan aman.
  - `validator_tool`: Membantu calon validator menghasilkan aset pendaftaran dan kunci-kunci yang diperlukan.
- **Utilitas CLI Zero-Knowledge:**
  - `generate_zk_params`: Alat sekali pakai untuk melakukan *trusted setup* dan menghasilkan *proving key* serta *verifying key* untuk sirkuit ZK-Rollup L2.

- `create_poseidon_params`: Alat sekali pakai untuk menghasilkan parameter untuk fungsi *hash* Poseidon yang digunakan di dalam sirkuit ZK.
- `prover`: Binary yang dipanggil oleh Sequencer untuk mengambil data *batch* transaksi dan menghasilkan bukti ZK-SNARK.
- `aggregator`: Mengambil beberapa bukti ZK dari *prover* dan menggabungkannya menjadi satu bukti agregat tunggal yang lebih efisien.

11 Kesimpulan

Evice Blockchain menghadirkan arsitektur canggih yang secara fundamental dirancang untuk mengatasi trilema blockchain. Dengan menggabungkan keamanan lapisan dasar yang diperkuat kriptografi *post-quantum* (Dilithium), finalitas absolut yang cepat melalui konsensus hibrida Aegis, dan skalabilitas masif dari ZK-Rollup yang terintegrasi, Evice siap menjadi platform pilihan untuk aplikasi terdesentralisasi yang menuntut kinerja tinggi, biaya rendah, dan keamanan tanpa kompromi.

Arsitektur modularnya, yang memisahkan eksekusi, konsensus, dan ketersediaan data, memberikan fondasi yang kokoh dan fleksibel. Penggunaan *runtime* WASM yang aman memungkinkan pengembangan *smart contract* yang kompleks dan efisien, sementara mekanisme ekonomi token yang seimbang dengan *fee burning* EIP-1559 dirancang untuk keberlanjutan jangka panjang. Kami percaya bahwa pendekatan berlapis, pemilihan teknologi yang berorientasi ke masa depan, dan ekosistem perkakas yang lengkap ini akan membuka jalan bagi gelombang inovasi berikutnya di ruang web3.

Pustaka

[1] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the weil pairing, 2001.

[2] Dhruba Borthakur, Siying Dong, Jason Kunnath, Murali Mullane, Mark Callaghan, and Nathan Bronson. RocksDB: A persistent key-value store for flash and ram storage. In *12th USENIX Conference on File and Storage Technologies (FAST '14)*, 2014.

[3] Vitalik Buterin, Eric Conner, Rick Dudley, Matthew Slipper, Ian Norden, and Abdelhamid Bakhta. EIP-1559: Fee market change. `https://eips.ethereum.org/EIPS/eip-1559`, 2019. [Online; diakses 21 Oktober 2025].

[4] Fatema David and Ian Goldberg. Schnorrkel: Schnorr signatures and vrfs on ristretto. Cryptology ePrint Archive, Report 2019/006, 2019. `https://eprint.iacr.org/2019/006`, [Online; diakses 21 Oktober 2025].

[5] Lorenzo Grassi, Dmitry Khovratovich, Christian Rechberger, Arnab Roy, and Markus Schofnegger. POSEIDON: A new hash function for zero-knowledge proof systems. In *30th USENIX Security Symposium*, 2021.

[6] Jens Groth. On the size of pairing-based non-interactive arguments. In *Advances in Cryptology – EUROCRYPT 2016*, pages 305–326. Springer Berlin Heidelberg, 2016.

[7] National Institute of Standards and Technology. FIPS 202: SHA-3 standard: Permutation-based hash and extendable-output functions. Technical report, U.S. Department of Commerce, 2015. [Online; diakses 21 Oktober 2025].

[8] National Institute of Standards and Technology. FIPS 204 (Draft): Module-lattice-based digital signature standard (ml-dsa). Technical report, U.S. Department of Commerce, 2023. [Online; diakses 21 Oktober 2025].

[9] NEAR Protocol. Borsh serialization format. `https://borsh.io/`, 2024. [Online; diakses 21 Oktober 2025].

[10] Yoav Nir and Adam Langley. ChaCha20 and Poly1305 for IETF Protocols. Request for Comments RFC 8439, Internet Engineering Task Force (IETF), June 2018. [Online; diakses 21 Oktober 2025].

[11] Parity Technologies. Parity db. `https://github.com/paritytech/parity-db`, 2023. [Online; diakses 21 Oktober 2025].

[12] Colin Percival and Simon Josefsson. The scrypt Password-Based Key Derivation Function. Request for Comments RFC 7914, Internet Engineering Task Force (IETF), May 2016. [Online; diakses 21 Oktober 2025].

[13] Protocol Labs. libp2p: The p2p networking stack. `https://libp2p.io/`, 2025. [Online; diakses 21 Oktober 2025].

[14] Serde Developers. Serde: A framework for serializing and deserializing rust data structures efficiently and generically. `https://serde.rs/`, 2025. [Online; diakses 21 Oktober 2025].

[15] Tokio Contributors. Tokio: An asynchronous runtime for the rust programming language. `https://tokio.rs/`, 2025. [Online; diakses 21 Oktober 2025].

[16] W3C WebAssembly Working Group. Webassembly core specification v2.0. `https://webassembly.github.io/spec/core/`, 2022. [Online; diakses 21 Oktober 2025].

[17] Wasmer Inc. Wasmer: The universal webassembly runtime. `https://wasmer.io/`, 2025. [Online; diakses 21 Oktober 2025].

[18] Gavin Wood. ETHEREUM: A secure decentralised generalised transaction ledger. Technical report, Ethereum Project, 2014. Gawain's Version, Berlin, [Online; diakses 21 Oktober 2025].

[19] Maofan Yin, Dahlia Malkhi, Michael K. Reiter, Guy Golan Gueta, and Ittai Abraham. HotStuff: BFT consensus in the lens of blockchain, 2019.