ChatGPT

IdenaAuthGo – Merkle Whitelist Generator

Overview: IdenaAuthGo is a Go-based backend tool that generates and serves *Merkle whitelist* data for Idena identities. It builds on Idena's Proof-of-Person (PoP) system to filter all **unique, validated human identities** in each validation epoch. The tool can authenticate users via the Idena app ("Login with Idena"), check eligibility criteria, and produce a whitelist of addresses that meet the PoP stake-and-status requirements. From this whitelist it constructs a Merkle tree: the `/merkle_root` endpoint returns the Merkle root, and `/merkle_proof?address=...` returns an inclusion proof for any address on the list [1] [2]. (These features mirror earlier Python prototypes, which likewise filtered out "bots" and low-stake/newbie identities [3] [1].) The primary use case is to **distribute on-chain rewards or grant group access** in a transparent, verifiable way (each user can check the Merkle proof against the published root). In short, IdenaAuthGo automates and documents the whitelist-generation process for Idena's anti-Sybil network [4] [1].

Key Features: The tool includes: - **Whitelist endpoints:** JSON APIs returning eligible addresses for the *current* or past epochs, and single-address checks. For example, `/whitelist/current` lists all eligible addresses for this epoch, and `/whitelist/check?address=<addr>` reports if an address is eligible right now [5] [2]. - **Eligibility logic:** Filters identities by Idena's PoP rules: only *Human*, *Verified*, or *Newbie* identities with sufficient stake are allowed. (Humans must meet the network's dynamic stake threshold, while Verified/Newbies need ≥10,000 iDNA [1] [3].) Identities with any **validation penalty** (e.g. caught cheating in the last session) are automatically excluded [6]. This matches the filtering used in the whitelist_blueprint Python script, which "filters out ... shitflippers, Humans below min stake, and Newbies/Verifieds below 10k IDNA" [3] [1]. - **Merkle proofs:** After building the whitelist, a Merkle tree is computed. The `/merkle_root` endpoint returns the root hash for the latest epoch, and `/merkle_proof?address=<addr>` returns the inclusion path if the address is on the list [6] [2]. The server can also be run in a snapshot mode (`go run main.go -index`), which manually regenerates the whitelist and prints the Merkle root to the console (saving it as `data/whitelist_epoch_<N>.json`) [7] [8]. - **Identity indexer:** A built-in "rolling indexer" continuously polls an Idena node and stores recent identity states in SQLite. It provides its own HTTP API (by default on port 8080) with endpoints like `/identities/eligible` and `/identity/{address}` [9] [10]. The main web server uses this local indexer (either via direct DB or HTTP) to answer whitelist and eligibility queries. The indexer uses **SQLite by default**, so no external database is needed [9] [11]. (You *don't* need the official Idena indexer/PostgreSQL; the built-in indexer covers all required data [11].) - **Agent scripts (optional):** For bootstrapping or debugging, there are helpers in `agents/`. For example, **identity_fetcher** polls a list of addresses and writes their identity snapshot to JSON (using a fallback public API if needed) [12] [13]. **session_block_finder** watches your node for when the validation ceremonies start, by detecting blockchain flags [14]. These agents are *not required* in normal use (the rolling indexer handles most needs), but can seed data or assist development.

Setup and Dependencies

- **Go & SQLite:** Install Go 1.20+ and ensure `sqlite3` is available. The project compiles to a Go binary and uses SQLite for its internal database.
- **Idena Node (local or remote):** You need access to an Idena full node (the official client). By default the code expects the node's RPC at `http://127.0.0.1:9009`. If you have a remote

node or custom port, set `RPC_URL` accordingly. The indexer polls the node's JSON-RPC for identity data, so the node must be *synchronized* with the network.

- **Environment variables:** Create a `.env` (or export in your shell) with at least:
- `BASE_URL`: the base URL where the backend will run (e.g. `http://localhost:3030`).
- `IDENA_RPC_KEY`: *(optional)* your node's RPC API key, if enabled.
- For the rolling indexer, set:
  - `RPC_URL` (your node's RPC URL, e.g. `http://localhost:9009`).
  - `RPC_KEY` (optional API key).
  - `FETCH_INTERVAL_MINUTES` (e.g. `10` to poll every 10 minutes).
  - `USE_PUBLIC_BOOTSTRAP` (`true` / `false` – see next point).
  - `BOOTSTRAP_EPOCHS` (e.g. `3` to pre-fetch last 3 epochs).
- **Public bootstrap (optional):** If your node is new or hasn't run for past epochs, set `USE_PUBLIC_BOOTSTRAP=true` in the indexer. This makes the indexer download historical identity data from a public source on first run, so it can "catch up" on missing epochs [15] [16]. This uses Idena's public APIs (e.g. the official idena-indexer API) as a fallback.
- **PostgreSQL (optional):** You generally *do not need* PostgreSQL. The built-in indexer uses SQLite by default. In contrast, the *official* Idena indexer uses PostgreSQL, but this project replaces that with SQLite to simplify setup [11].
- **Python (optional):** A separate Python helper `whitelist_blueprint` is available for one-off whitelists. It requires Python 3.10+ and the `requests` package, and can fetch data directly via Idena's API [17]. But in normal operation, you only need the Go code.

Running the Service

1. **Clone and configure:**

```
git clone https://github.com/ubiubi18/IdenaAuthGo.git
cd IdenaAuthGo
cp .env.example .env      # then edit .env as described above
```

Ensure `BASE_URL` and (if needed) `IDENA_RPC_KEY` are set.

2. **Start the web server:**

```
go run main.go
```

This launches the API on port 3030 (or your `BASE_URL`). It will automatically rebuild the whitelist and Merkle root after each validation epoch.

3. **(Recommended) Run the rolling indexer:**
In a separate terminal, build and launch the indexer service which feeds identity data to the web server:

```
cd rolling_indexer
go build -o rolling-indexer main.go
export RPC_URL="http://localhost:9009"          # your Idena node RPC
export RPC_KEY="..."                            # node API key if any
```

```
export FETCH_INTERVAL_MINUTES=10
export USE_PUBLIC_BOOTSTRAP=true              # fetch past data
export BOOTSTRAP_EPOCHS=3
./rolling-indexer
```

By default, the indexer creates `rolling_indexer/identities.db` (SQLite) and listens on port 8080. Test it with, e.g.:

```
curl http://localhost:8080/identities/eligible
```

You should see a JSON array of currently eligible addresses [9] [11].

4. **Optional agents:** The `agents/identity_fetcher.go` can be used to manually fetch selected addresses (see *AGENTS.md*). The `agents/session_block_finder.go` can detect when validation sessions start. These are not needed for normal use but can help in custom scenarios [18] [14].

API Endpoints

Once running, IdenaAuthGo provides the following HTTP APIs (JSON):

- **POST /signin**, **GET /callback:** (Optional) Initiate and handle a "Login with Idena" deep-link flow for user authentication.
- **GET /eligibility?address=<addr>:** Returns the identity status and stake for `<addr>` at the latest snapshot, indicating if it meets PoH criteria.
- **GET /whitelist/current:** Returns a JSON list of all eligible addresses (and their statuses) for the current epoch.
- **GET /whitelist/epoch/{epoch}:** Returns the whitelist for a past epoch number.
- **GET /whitelist/check?address=<addr>:** Checks if `<addr>` is on the current whitelist (and why or why not).
- **GET /merkle_root:** Returns the hex Merkle root of the current whitelist (epoch).
- **GET /merkle_proof?address=<addr>:** If `<addr>` is in the whitelist, returns the Merkle proof (as a list of hashes) verifying its inclusion; otherwise an error.

Example usages (using `curl`):

```
curl "http://localhost:3030/eligibility?address=0xYourAddress"
curl "http://localhost:3030/whitelist/current"
curl "http://localhost:3030/merkle_root"
```

These return JSON. For example, `/merkle_root` might return:

```
{"merkle_root":"0xabcdef1234..."}
```

or you can use `go run main.go -index` to manually generate and print the root [7] [8].

Developer Notes

- **Whitelist generation:** Internally, the server recomputes the whitelist at epoch boundaries. It queries the rolling indexer or node RPC for all identities, filters by state (Human/Verified/Newbie), minimum stake, and excludes any with penalties [1] [3]. It then writes `data/whitelist_epoch_<N>.json` and builds a Merkle tree from that sorted list.
- **Indexing logic:** The built-in indexer continuously polls the node for identity updates (via RPC). You *can* skip running it by using the `identity_fetcher` agent to collect data, but the indexer is the intended data source. The indexer uses an in-memory rolling window (≈30 days) of identity history and serves queries to the main server [9] [11].
- **Configuration:** All major settings (RPC URLs, API keys, stake thresholds, etc.) can be adjusted via the `.env` or JSON config files. Review the examples in the repository (`.env.example`, `rolling_indexer/config.json.example`, `agents/*.example.json`) for guidance.
- **Rebuilding snapshot:** To force a manual whitelist recomputation (for testing), stop the web server and run `go run main.go -index`. This will use the indexer's DB or the node's RPC (if no indexer) to fetch identity data and output the Merkle root, as noted above [7] [8].
- **Monitoring:** Logs are printed to stdout. You can also query the indexer's endpoints (e.g. `/identities/eligible`) to inspect which addresses it considers eligible [9].

Frontend/UI Guidance

To build a simple web UI (using React, Vue, or plain JS), consider the following design and behavior:

- **Title:** At the top, display **"Idena Eligibility Discriminator – Generate Whitelist Merkle Root"**.
- **Mode Buttons:** Provide two buttons side-by-side:
- **"From your own node"** – triggers generation using your connected Idena node (via the local indexer).
- **"From the public indexer"** – triggers generation by calling a public Idena API (e.g. https://api.idena.io) without requiring a local node.
- **Live Log Panel:** Include a scrollable console/log area. When the user clicks a button, start the generation process and stream output lines here (e.g. "Fetching identities...", "Filtering 500 -> 230 eligible...", "Computing Merkle root...", etc.). You can use WebSocket or server-sent events to show real-time progress from the backend logs.
- **Result Display:** When done, show the final Merkle root in a text box with a "Copy" button. Also indicate which epoch it corresponds to. The user should be able to easily copy the root hash.
- **Address Checker:** Below, add a section "Check Address" with an input field and button. When the user enters an address and clicks "Check", call `/whitelist/check?address=...` (or the public API) and display whether it is eligible or not, along with its identity status and stake. If eligible, optionally fetch and display the Merkle proof via `/merkle_proof` so they can verify inclusion.
- **Developer Hints (Codex prompts):** Explain that clicking either button runs the same Merkle generation logic, just with different data sources. Emphasize streaming logs (e.g. "use `eventsource` or websockets for live updates"). Show the Merkle root prominently. The UI should be clean and minimal. You might prompt: *"Create a React component with state for `merkleRoot`, `logs` and `loading`. Use two buttons that call the backend via fetch/AJAX. Append each log message to `logs`. When complete, update `merkleRoot`. Also include an address input and onSubmit fetch eligibility and proof."*

FAQ

- **Why use Idena?** Idena is a **Proof-of-Personhood** blockchain: it grants each human one unique "cryptoidentity" without using personal data [19] [20]. Every participant must solve synchronous puzzles (FLIP tests) in validation ceremonies. These puzzles are easy for humans but hard for AI, ensuring uniqueness. Idena's goal is to give every verified human one voice and one vote (and one share of the coin issuance), preventing Sybil attacks [19] [21]. The whitelist generator builds on this by selecting only the identities that remain *currently valid* (no skipped validations, enough stake, etc.) for use in airdrops, votes, or group access.

- **How does PoP (Proof-of-Personhood) work?** Idena's PoP protocol uses **synchronous Turing tests**: all candidates solve 6 short "FLIP" puzzles in under 2 minutes [22] [21]. Uniqueness is ensured by this tight timing: a human cannot be in two places at once, so one person can't validate multiple accounts simultaneously [23] [21]. After each epoch, only those who passed remain "human" or "verified" until the next session. No identity (or stake) can be reused or sold, because selling one's private key destroys trust – the seller could always kill the identity later [24] [25]. In short, Idena avoids central ID checks by having people test each other's humanness and change identities every epoch [19] [21].

- **What about AI or farming attacks?** The core assumption is that FLIP puzzles are currently **hard for AI**. In fact, a Stanford study notes "Idena's FLIPS were successfully AI-resistant, where bots did not succeed in generating fake accounts" (2019–2022) [26]. Ongoing efforts like the Idena Flip Challenge encourage building open AI solvers, but so far human-level general intelligence is needed to understand the puzzles. This follows the "easy for humans, hard for AI" principle of modern benchmarks (see the ARC-AGI challenge) [27]. Even if an AI could occasionally solve a flip, it would need an enormous network and coordination to farm many identities per epoch, which quickly becomes infeasible. Moreover, any systematic abuse (e.g. coordinated chains of invitees) risks detection and community pushback. The system also uses stake incentives and locked coins to discourage selling or mass-inviting accounts [24] [25]. In essence, proof-of-human-work (PoH) protocols like Idena rely on tasks that are verifiably performed by a unique human. Academic definitions of PoH note that such puzzles are *"moderately hard for a human"* but *"hard for a computer to solve… without sufficient assistance from a human"* [28]. As AI evolves, these tests will need updates, but currently Idena's approach holds up under review [26] [27].

- **Why is this important?** Ensuring one-account-per-human is critical for fair governance, universal basic income distribution, and censorship-resistant voting. Decentralized systems like Idena address the growing risk of automated bots and fake identities online [20] [27]. By combining cryptography with human verification, we create a foundation for a more equitable digital society. (For broader context, see Idena's [official FAQ/docs] and research on Proof-of-Personhood [19] [26]. The ARC and ARC-AGI projects also illustrate why intuitive, human-style reasoning tasks remain a strong barrier for bots [27] [28].)

Contributing

IdenaAuthGo is open-source and welcomes contributions. Please review the code and documentation on GitHub. You can submit issues or pull requests to help improve features or fix bugs. We encourage clear documentation, unit tests, and active discussion to keep the whitelist logic robust and transparent. Since this is a public verification tool, please ensure any changes preserve the security properties (e.g. correct handling of penalties and stake thresholds). Community feedback on new use cases or UI/UX improvements is especially welcome.

References: Official Idena FAQ/whitepaper and documentation [19] [21] describe the PoP protocol. Research like *"Compressed to 0"* (Stanford) reviews Idena's Proof-of-Personhood [26]. Proof-of-human-work is discussed in academic literature [28]. The ARC-AGI challenge emphasizes tasks easy for humans, hard for AI [27]. These sources, among others, provide deeper context on the principles behind Idena and the whitelist approach.

---

[1] [2] [5] [6] [7] [8] [9] [10] [11] [12] [15] README.md
https://github.com/ubiubi18/IdenaAuthGo/blob/d0be273246054211a8329deeaf2db970189111ab/README.md

[3] [4] [17] README.md
https://github.com/ubiubi18/whitelist_blueprint/blob/793441a12cf9bc89b52455f8605755b129a06313/README.md

[13] [14] [16] [18] AGENTS.md
https://github.com/ubiubi18/IdenaAuthGo/blob/d0be273246054211a8329deeaf2db970189111ab/AGENTS.md

[19] [23] [24] Idena Whitepaper - Technology | Idena documentation
https://docs.idena.io/docs/wp/technology

[20] Proof of human is essential, and it's going mainstream in 2025
https://world.org/blog/world/proof-of-human-essential-going-mainstream-2025

[21] [25] IDENA FAQ
https://www.idena.io/faq

[22] [26] 2024-compressed_to_0.xopp
https://ia601600.us.archive.org/35/items/elopio-papers/2024-compressed_to_0-annotated.pdf

[27] ARC Prize - What is ARC-AGI?
https://arcprize.org/arc-agi

[28] eprint.iacr.org
https://eprint.iacr.org/2016/145.pdf