

Giver.EU: Sealed Editions

End-to-End Encrypted Exchange of Digital Goods
with Deadline-Bound Escrow on Solana

Greenpaper v1.2

Prospero

Guild of Exchange & Transport of Kyberia

October 2026

Abstract

Giver.EU is the European peer-to-peer bazaar of knots, goods and services built on Solana, born in the Kyberia community, made in the European Union, spoken in its 24 official languages and rewarding in its own currency, Kybchain (unit: GANESHA). Version 0.1 of this paper described an escrow for physical goods, released by the joint attestation of giver and redeemer. This paper describes the second layer: *sealed editions*, a protocol for giving information in exchange for a reward—texts, PDF bundles, datasets, source code, software licence keys, passwords—to an exact, giver-chosen number of people, such that *the bazaar operator never learns the goods, the keys or the secrets it relays*. Content is encrypted in the giver’s browser with a random content key; each redeemer receives that key inside an individual HPKE envelope addressed to an encryption key their wallet has signed for. The redeemer’s reward is held by an on-chain escrow program that releases funds only after the giver commits to a delivery, and refunds the redeemer automatically when a fulfilment deadline passes. Scarcity is enforced by a fixed-supply Token-2022 edition mint; reputation emerges from on-chain-verifiable delivery events. One rule governs ownership: *the knot is the key*—whoever holds an edition can open it, and when it is passed on, the holder’s browser re-encrypts the content key to the next holder while the token moves on chain. We describe the construction, its security properties and limits, the cold-key configuration prepared for mainnet, and report an implementation that is deployed on Solana devnet and tested end to end.

1. Introduction

Fifteen years after the genesis of the planetary elliptic-curve discrete-logarithm bounty programme, cryptocurrencies still lack an everyday market in which ordinary people exchange ordinary things. Version 0.1 of Giver.EU [1], coded in eight days during the Colosseum Cypherpunk hackathon, set out to build such a market for the oldest Slovak digital community, kyberia.sk [2], whose internal content economy predates Bitcoin [3]. Its escrow fitted physical goods: the redeemer deposits the reward, the parties meet, both sign an attestation, the vault releases it.

Information behaves differently. It is copied at no cost, it is delivered at a distance, and its giver is often a single author who wants to share a text with *exactly* two colleagues, a code bundle with five, a licence with seven. Three requirements follow:

1. **Scarcity:** the number of copies must be fixed in advance and visible to everybody.
2. **Fair exchange at a distance:** the redeemer must not lose the reward if the giver never delivers; the giver must not deliver without being rewarded.
3. **Confidentiality towards the platform:** the operator must not be able to read, leak or resell what passes through it. For licence keys and passwords this is the whole point; for unpublished writing it is a precondition of trust.

Giver.EU answers these with three mechanisms, each independent of the others: a fixed-supply edition mint (Section 2), an end-to-end encryption protocol (Section 3) and a deadline-bound escrow program (Section 4).

Roles. The *giver* presents an item; the *redeemer* acquires it; the *vault* (now: the escrow program) holds value under conditions; the *arbiter* resolves disputes the program cannot decide; the *crank* is any party that triggers a time-based settlement. The Giver.EU server only stores ciphertext, relays messages and runs the crank.

A European bazaar. Giver.EU is operated by wizzion.com UG (Berlin) and speaks all 24 official languages of the European Union; the interface follows the visitor’s browser or an explicit choice, remembered locally and, after signing in, with the wallet. It loads no third-party resources (fonts, scripts and images are

self-hosted), sets no tracking cookies and implements the notice-and-action duties of the Digital Services Act (Section 11).

2. Editions

Every digital listing is a set of *knots* (**KN**owledge **O**wnership **T**okens): a Token-2022 mint with decimals 0, an on-chain metadata extension (name, symbol, URI) and a supply of exactly N tokens minted to the giver. In the same transaction the giver approves the marketplace delegate for N tokens and revokes the mint authority, so the supply is provably fixed forever. Publishing costs one wallet signature and ≈ 0.005 SOL of rent.

Giver.EU supports two delivery modes on top of an edition:

Instant. For non-secret goods, reward and edition are exchanged atomically: the server builds one transaction (redeemer $\rightarrow$ giver: reward-fee; redeemer $\rightarrow$ treasury: fee; giver $\rightarrow$ redeemer: one edition, moved by the delegate) and co-signs it only as delegate. Either everything happens or nothing; overselling is impossible because the supply is on chain. Files are hosted by Giver.EU.

Sealed. For secrets and anything the platform must not see; the subject of the rest of this paper. The edition is handed to the redeemer by the crank once the reward has been released.

The knot is the key. Both modes follow one rule: *whoever holds an edition can open it*. In instant mode the server serves the files to the wallet holding the token (a purchase receipt opens them only while the redeeming wallet still holds it). In sealed mode the content key travels with the token: passing a knot on re-encrypts the key to the next holder in the current holder’s browser (Section 3.7). Licence keys follow the holder in both modes. Whoever passes a knot on loses access *through Giver.EU*; nobody can be made to forget a file they have already saved, just as a book that has been read and given away.

Goods and services. The same escrow program settles any physical good or service. An offer is published without any transaction: photographs (re-encoded, all metadata including GPS position stripped), a description that may be dictated and transcribed by a faster-whisper speech-to-text service, a reward and a quantity. The redeemer commits the reward to the escrow and sends a shipping address or appointment note end-to-end encrypted to the giver (HPKE, context GIVER-MESSAGE-v1); the giver marks the order delivered by committing the hash of an encrypted delivery note (tracking code, pickup time); the redeemer confirms receipt, or the reward is released after the confirmation window. The joint-attestation flow of v0.1 is thereby replaced by a single generic primitive.

3. Sealed delivery

3.1 Goals and threat model

The Giver.EU server, its operators and anyone who compromises them are treated as honest-but-curious at best and malicious at worst. They must never obtain: the plaintext goods, the content key K , any per-redeemer secret, or any private encryption key. They may observe metadata: sizes, timing, wallet addresses, reward amounts. A malicious server may drop, replay or substitute messages; such attacks must be *detected* by the redeemer. No cryptographic primitive is invented: we compose well-reviewed constructions with audited implementations.

3.2 Primitives

- Asset encryption: XChaCha20-Poly1305 (192-bit random nonce, 256-bit key), implementation `@noble/ciphers`.
- Key encapsulation and envelopes: HPKE, RFC 9180 [4], suite DHKEM(X25519, HKDF-SHA256) / HKDF-SHA256 / ChaCha20-Poly1305, implementation `@hpke/*` (pure-JavaScript X25519, so no dependence on browser WebCrypto support). Envelopes use `mode_auth`, which authenticates the sender’s static key.
- Hashing: SHA-256. Wallet signatures: Ed25519 (the Solana account key).

The Solana wallet authenticates and signs the exchange; it is *not* used as an encryption key. Encryption keys are separate X25519 key pairs generated in the browser.

3.3 Encryption identities

A wallet binds an X25519 public key to itself by signing a domain-separated statement that authorises nothing else:

```
GIVER-ENCRYPTION-KEY-v1
wallet: <base58 wallet address>
encryption-key: <base64 X25519 public key>
key-version: <n>
domain: giver.eu
issued-at: <ISO-8601 UTC>
```

I bind this X25519 encryption key to my wallet for end-to-end encrypted deliveries on Giver.EU.
This signature authorizes no payment, login or any other action.

The server verifies the exact format, the domain, freshness (± 15 min), the Ed25519 signature and that the version is the successor of the last one, then stores the binding. It never accepts an unverified key. Identities are versioned: a replacement key is registered by signing a new binding; past purchases stay bound to the key they were delivered to and are never silently re-encrypted. Private keys live in the browser only; users can export a backup encrypted with a passphrase (scrypt $N=2^{15}$, XChaCha20-Poly1305). Losing an unbacked key makes content delivered to it unrecoverable—by design, nobody can recover it on the user’s behalf.

3.4 Content encryption

When listing, the giver’s browser draws $K \xleftarrow{\$} \{0,1\}^{256}$ and encrypts each part i of the goods (a file, or a text secret shared by all redeemers) as

$$C_i = \text{"GVE1"} \parallel n_i \parallel \text{XChaCha20Poly1305}_K(n_i, \text{len} \parallel \text{hdr} \parallel m_i; \text{AAD}_i), \quad \text{AAD}_i = \text{GIVER-ASSET-v1} \parallel \text{listing} \parallel i,$$

where *hdr* carries the original file name and media type, so that even names are hidden from the server. PDFs are treated as opaque bytes; no reliance is placed on PDF password protection. Only the C_i are uploaded. The server checks that every part carries the magic prefix (so plaintext cannot be uploaded by mistake) and recomputes the asset hash

$$H_A = \text{SHA256}(\text{GIVER-ASSET-MANIFEST-v1} \parallel (i : \text{SHA256}(C_i) : |C_i|)_i),$$

which must equal the value computed by the giver. Before encryption the browser also extracts public *document facts* (size, PDF page count or word count, and the SHA-256 of the plaintext), shown on the knot page so that redeemers can verify the decrypted file. One ciphertext serves all redeemers; it is never re-encrypted per redeemer.

Giver.EU vault. So that the giver can fulfil later and from any device holding their key, K —and, for licence listings, the list of per-edition secrets—is sealed with HPKE base mode to the giver’s *own* encryption key, with *info* binding listing, giver wallet and key. The server stores this vault but cannot open it.

3.5 Key envelopes

For a redemption p by redeemer b of listing ℓ from giver g , the giver’s browser opens its vault and computes

$$E_p = \text{HPKE.Seal}_{\text{auth}}(pk_b, sk_g; \text{info}=\text{ctx}, \text{aad}=\text{ctx}; \{v, \ell, p, b, g, H_A, K, s_p\}),$$

where s_p is the redeemer’s personal secret (e.g. licence key) if any, and the context string is

```
GIVER-ENVELOPE-v1
protocol: GIVER-E2E-v1
listing: <id> purchase: <id>
buyer: <wallet> seller: <wallet>
asset: <H_A>
recipient-key: <pk_b> sender-key: <pk_g>
```

(one field per line in the implementation). The binding prevents replay into another purchase, substitution of another asset, re-targeting to another redeemer and confusion between key versions; *mode_auth* additionally lets the redeemer reject envelopes not produced by the giver’s registered key—including any envelope forged by the server. After decryption the redeemer compares every payload field with the expected context. The server only ever stores and relays E_p .

3.6 Redemption

The redeemer fetches E_p and the C_i , opens the envelope with sk_b , checks the bindings, verifies each part hash and H_A , then decrypts each part locally. Every failure mode—wrong key, tampered envelope, wrong asset, truncated or modified part—is detected by an AEAD tag or a hash comparison; no plaintext is ever produced from manipulated input.

3.7 Passing a knot on

Let h be the current holder of edition p (initially the redeemer b) and r the recipient, who has registered an encryption key pk_r . The holder’s browser opens its own envelope (from the giver, or from the previous holder), recovers K and the personal secret s_p , draws a 128-bit pass id π and seals

$$P_\pi = \text{HPKE.Seal}_{\text{auth}}(pk_r, sk_h; \text{info}=ctx_\pi, \text{aad}=ctx_\pi; \{v, \ell, p, \pi, h, r, g, H_A, K, s_p\})$$

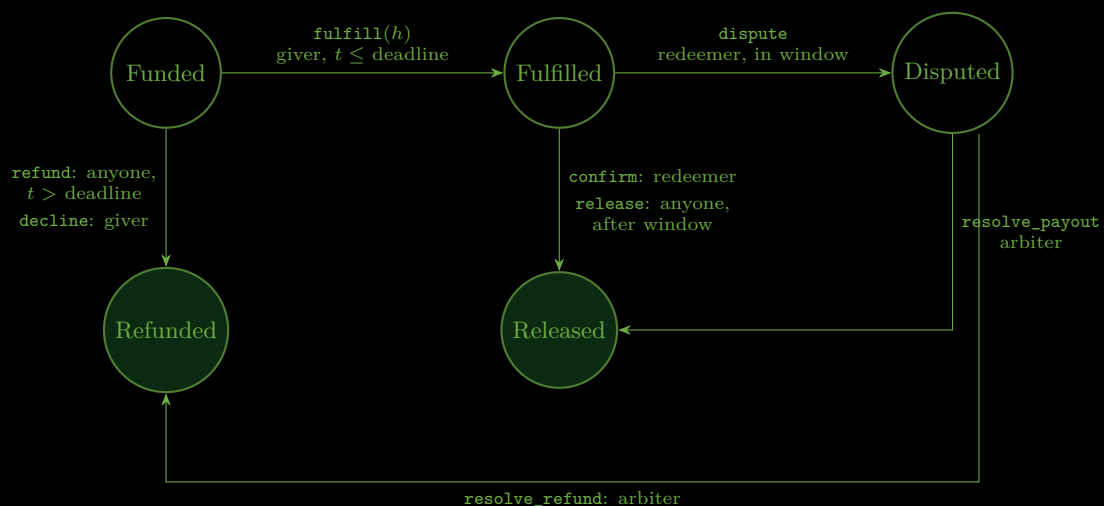
with the domain-separated context

```
GIVER-PASS-v1
protocol: GIVER-E2E-v1
listing: <id> purchase: <id> pass: <pi>
from: <holder wallet> to: <recipient wallet> seller: <giver wallet>
asset: <H_A>
recipient-key: <pk_r> sender-key: <pk_h>
```

The server stores P_π as *pending* and returns one transaction for the holder to sign: create the recipient’s token account if needed, `transferChecked` one edition from holder to recipient, and a memo `giver:pass: $\pi$` . Access changes hands only after the server has found that transfer on chain—exact mint, source, destination, authority, amount 1 and memo; the hand-over is a compare-and-set on the current holder, so a knot cannot be passed twice. If the holder’s page closes before confirming, a worker recovers the pass by searching the recipient’s token account for the memo. The recipient opens P_π with sk_r , checks every bound field and that the sender key belongs to the previous holder, and decrypts the unchanged ciphertext. A pass can neither be replayed into another pass, redirected to another wallet, opened by a third party nor mistaken for a delivery envelope. Passing on requires no action by the giver, and Giver.EU still never sees K . A token sent with a wallet application instead of through Giver.EU moves no key; users are told to pass knots on through Giver.EU.

4. Escrow program

The deployed v0.1 vault could only release on joint attestation and had no notion of time, so a silent giver would lock the redeemer’s money forever. Sealed redemptions therefore use a new Anchor program, `giver_escrow` (7Yq7PSKmsrPenCHhNBu8pV5897oheTrnSj12ow14vXC6 on devnet). One escrow account, seeded by a random 16-byte purchase id, records buyer, seller, mint, amount, deadline, dispute window, the fulfilment time, an envelope commitment and a state. Funds sit in the escrow PDA’s associated token account. (On chain the giver is stored as `seller` and the redeemer as `buyer`.)



Release sends $\text{amount} - \text{fee}$ to the giver and fee (1 %) to the treasury; refunds return the full amount. Escrow and vault rent (≈ 0.004 SOL) always returns to the redeemer, who covered it. The program is token-agnostic (SPL Token and Token-2022 via the token interface); rewards on the bazaar are denominated in GANESHA.

What the program does not know. The program cannot tell whether K is correct: it never sees K , and verifying it would require decrypting. What it does record is a public, timestamped commitment to

exactly which envelope the giver delivered, which (i) prevents a giver from later claiming a different delivery, (ii) lets the server verify that the stored envelope is the committed one, and (iii) gives the redeemer the dispute window to object. The commitment, the AEAD tags and the asset hash together provide “enough authenticated metadata to detect malformed or incorrect decryption” without the operator ever obtaining the key.

Timeouts. A worker mirrors escrow state and cranks `refund` once the on-chain deadline passes and `release` once the dispute window closes. Both instructions are permissionless; if Giver.EU disappears, anybody can crank them, so funds are never stranded.

Keys on mainnet: vanity where possible, cold wherever possible. The arbiter and the treasury are compiled into the program, so each cluster has its own build; a build script substitutes the three constants and writes a manifest that a read-only preflight compares with the configuration and, after deployment, with the deployed code.

Role	Mainnet key	Custody
Escrow program	<code>GiveUoTgYLcmC14PuWEciMYnmLGDEPJpXtyZnbNbdRR5</code>	key needed only to deploy
Arbiter, treasury, upgrade authority	<code>GVR3dJHZJ8vQkr7N9auKq5fqcms61G4Tum2NjMb2Qc4u</code>	cold, ground offline; never on the server
Crank	<code>GVRhUbdG...5wq9</code>	hot; pays settlement fees, holds ≈ 0.3 SOL
Delegate	<code>GVRBBPgK...cEFd</code>	hot; co-signs instant edition swaps, holds no funds
Knot mints	<code>Give...</code>	pre-ground pool; each key used once, powerless afterwards

The arbiter decides a dispute by connecting its wallet to the administration page and signing one `resolve` transaction; no message signature (which hardware wallets often cannot produce) is required, since only the compiled-in key can make the decision effective. The devnet deployment (`7Yq7...vXC6`) keeps its own test keys.

5. Fees and anti-spam

Presenting a knot or an offer costs an *upload fee* of 1 GANESHA per started megabyte of uploaded files or photographs (minimum 1 GANESHA), transferred to the treasury inside the very transaction that publishes the knot; the server activates a listing only after verifying that transfer on chain (exact amount, giver as authority, treasury account, listing memo). Completed exchanges carry a 1% fee enforced by the escrow program; returned rewards are free. Until the program has been externally audited, a single redemption may lock at most 10^6 GANESHA in escrow; the cap is enforced when a knot is presented or edited and bounds what an undiscovered defect could affect. Giver.EU frames every exchange as a reward between peers: it is a bazaar for exchange and barter, not an officially recognised marketplace or merchant, and it never holds the rewards itself.

6. Trust and reputation

Every state change emits an append-only trust event: `delivered`, `completed`, `declined`, `disputed`, `fulfillment_timeout`. Events are facts derivable from the chain; the score is a pure, swappable function of them. The current default is

$$S(w) = \sum_{e \in E_w, e \neq \text{timeout}} \omega(e) + \sum_{k=0}^{T_w-1} \omega_{\text{timeout}} (1 + 0.5^k),$$

with $\omega = \{\text{delivered}: 1, \text{completed}: 0.5, \text{declined}: -0.5, \text{disputed}: -1, \text{timeout}: -3\}$, so each additional missed deadline weighs 50 % more than the previous one. Labels (*new*, *reliable*, *trusted*, *mixed*, *unreliable*) are shown on every sealed listing. Weights are configuration, not code.

7. Transparency and social signals

Every knot page shows its public redemption history—edition, redeemer (alias and identicon), time, reward actually escrowed and outcome (in escrow, delivered, disputed, completed, refunded), each linked to the settling transaction or escrow account. Nothing here is secret: wallets, amounts and signatures are on chain anyway. Knot pages also count unique visitors: a visit is recorded once per visitor and day under a keyed, daily-rotating hash of address and browser, so no IP address is stored and no visitor can be followed across days; known crawlers are ignored. The counts are public, allow sorting the bazaar by “trending today” and “most viewed”, and give redeemers an honest sense of interest (“5 people looked at this today, only 2 of 7 left”).

8. Licences

A licence listing carries N secrets, generated locally (`GIVR-XXXXX-XXXXX-XXXXX-XXXXX`, ≈ 100 bits) or pasted by the giver, stored only inside the giver vault. At fulfilment the next secret is placed in the redeemer’s envelope. If the giver enables verification, the browser uploads only domain-separated hashes `SHA256(GIVER-LICENSE-v1|| $\ell$ || $s$ )`; the public endpoint `GET /api/license/{id}/key/{key}` answers whether a key is genuine, redeemed and not under dispute, and to which wallet—without the operator ever holding the key.

Ownership semantics. A licence follows the knot. Passing a sealed licence knot on carries the licence key inside P_π , and the verification endpoint then names the new holder; instant licence keys are valid only while their wallet holds the token and are re-bound by the next holder. Version 1.0 bound sealed access to the purchasing wallet and announced transferable licences as explicit re-keying; Section 3.7 delivers that re-keying without involving the giver.

9. Wallets and usability

Giver.EU discovers every Wallet-Standard wallet (Phantom, Solflare, Backpack, OKX, Coinbase, ...), registers the Mobile Wallet Adapter on Android, deep-links into wallet browsers on iOS and offers Solana Pay transaction requests (QR) to desktops without an extension. Every transaction is simulated server-side before it reaches the wallet, so users see “you need 0.05 GANESHA” rather than a cryptic failure. Givers set an alias and an avatar; wallets without one get a deterministic pixel identicon. On devnet a faucet mints test GANESHA and a little SOL. Listing is as short as on any centralised marketplace: drop files, choose the number of editions and a reward, sign once.

10. AI agents

Giver.EU is usable by AI agents on the same terms as by people: reading is open, and nothing that moves value happens without the user’s own wallet signature. Three interfaces serve agents (details in Annex A):

- a remote *Model Context Protocol* server [9] at `/api/mcp` with read-only tools (search, knot details, redemption history, statistics, licence verification, FAQ);
- *WebMCP* [10] in the browser: every page registers tools through `document.modelContext` (or `navigator.modelContext` in earlier drafts), so an agent acting for the user can search, navigate, fill in the give form, or *start* a redemption or a pass-on—the wallet then asks the human to approve;

self-dealing), also as the exact mainnet binary, where a key other than the compiled-in arbiter is refused; the protocol passes 27 tests including replay, substitution, wrong recipient, server-forged envelopes and the pass cases above, also in browsers lacking WebCrypto X25519; browser end-to-end tests against Solana devnet cover publish, key registration, escrow, fulfilment, byte-identical decryption, early release, licence verification, deadline refund by the crank and disputes (28 checks), and a second run of 30 checks adds self-hosted software, passing a knot on to a third wallet (token moved, new holder decrypts and inherits the licence, previous holder locked out), a dispute decided by the arbiter wallet in the administration page, and a content report.

14. Roadmap

- Mainnet launch with Kybchain GANESHA; the first knot is `giver.tgz`.
- External audit of `giver_escrow`, then lifting the beta cap; a multisig for the cold roles.
- Re-keying for tokens moved outside Giver.EU (by the giver’s vault or a Token-2022 transfer hook); a secondary market for knots.
- Stablecoins and other SPL tokens as reward currencies; other Giver.EU instances may use their own utility token.
- Native-speaker review of all 24 languages.
- Arbitration by the community rather than a single key—e.g. panels drawn under Kyberia’s Parallel Democracy model [2]—with the arbiter replaced by a multisig or a programmatic vote.
- Content on IPFS/Arweave addressed by H_A .
- Giver-side delivery agents so that givers need not be online, without giving up end-to-end encryption.

Rights

All rights to the innovations and the code of Giver.EU belong to wizzion.com UG (haftungsbeschränkt) and the Guild of Exchange & Transport of Kyberia. Support and future updates of Giver.EU are provided to the owners of the `giver.tgz` knot (<https://giver.eu/k/2-giver-tgz>).

Changes in v1.2 (greenpaper): AI agents (Section 10, Annex A); remote MCP server, WebMCP browser tools, `llms.txt` and OpenAPI; knots are *Knowledge Ownership Tokens*; set in Kyberia green.

Changes since v1.0: the knot is the key and passing knots on (3.7); cold-key mainnet configuration and beta cap (4); redemption history and view counts; compliance and operations (11); 24 languages.

Acknowledgements

To the kyberians who kept a giving economy alive for a quarter of a century, to the authors of v0.1, and to every giver who trusts a stranger with something they made.

References

- [1] D. D. Hromada and the Guild. *Giver: Toward an On-Chain Escrow Marketplace of Tangible and Intangible Assets*, Whitepaper v0.1, October 2025. <https://giver.eu/whitepaper-v0.1.pdf>
- [2] D. D. Hromada. Parallel Democracy Model. Kyberia, 2013.
- [3] D. D. Hromada. Kyberia Open Ledger. 2022.
- [4] R. Barnes, K. Bhargavan, B. Lipp, C. Wood. *Hybrid Public Key Encryption*. RFC 9180, IETF, 2022.
- [5] S. Arciszewski. *XChaCha: eXtended-nonce ChaCha and AEAD_XChaCha20_Poly1305*. IRTF CFRG draft.
- [6] Solana Labs. *Token-2022 Program and Token Metadata Interface*. <https://spl.solana.com>
- [7] Wallet Standard; Solana Mobile Wallet Adapter; Solana Pay specification.
- [8] S. Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System*. 2008.
- [9] Model Context Protocol, specification 2025-06-18 (Streamable HTTP transport). <https://modelcontextprotocol.io>
- [10] W3C Web Machine Learning Community Group. *WebMCP* (draft). <https://github.com/webmachinelearning/webmcp>
- [11] J. Howard. *The /llms.txt file*. <https://llmstxt.org>

A. Annex: routes and APIs for AI agents

A.1 Remote MCP server

Endpoint	POST https://giver.eu/api/mcp (JSON-RPC 2.0; GET answers 405: stateless, no server stream)
Transport	MCP Streamable HTTP, JSON responses; protocol versions 2025-06-18, 2025-03-26, 2024-11-05
Methods	<code>initialize</code> , <code>ping</code> , <code>tools/list</code> , <code>tools/call</code> ; notifications answer 202; JSON-RPC batches accepted
Errors	tool failures are results with <code>isError: true</code> ; unknown tools —32602, malformed JSON —32700
Limits	per-IP rate limits of the API (10 reads/s)

All tools are read-only (`readOnlyHint`) and return `content` (JSON text) plus `structuredContent`:

Tool	Arguments	Result
<code>search_knots</code>	<code>query?</code> , <code>kind?</code> (all, digital, goods_or_service), <code>sort?</code> (new, trending, views), <code>limit?</code> (1–60)	knots: id, title, teaser, kind, giver, reward (and ladder), editions total/left, views, URL
<code>get_knot</code>	<code>id</code>	the above plus description, document facts (size, pages, SHA-256), edition token mint, licence flag
<code>get_redemptions</code>	<code>id</code>	edition, redeemer, time, reward, outcome, transaction or escrow account
<code>bazaar_stats</code>	—	live knots, redemptions, givers; latest redemptions
<code>verify_licence_key</code>	<code>knot_id</code> , <code>key</code>	valid, reason, edition, holder (licences follow the knot)
<code>faq</code>	<code>query?</code>	matching questions and answers
<code>about_giver</code>	—	network, currency mint and decimals, escrow program, fee, beta cap, links

A.2 WebMCP in the browser

The module `/js/app/agent.js` detects `document.modelContext` (current draft) or `navigator.modelContext` (earlier drafts) and registers tools with `registerTool(tool, {signal})`, falling back to `provideContext({tools})`. Without either it does nothing. Read tools forward to the MCP server above, so both interfaces return identical data. Results follow the draft's `{content: [{type: "text", text}]}`.

Where	Tool	Effect
every page	the seven read tools	as on the MCP server
every page	<code>open_page</code>	navigate to bazaar, a knot, give, offer, library, FAQ, video, newsletter, greenpaper
every page	<code>set_language</code>	switch among the 24 EU languages
every page	<code>play_demo_video</code>	play the two-minute demo in an overlay
every page	<code>report_knot</code>	send a Digital Services Act notice (user's explicit request only)
every page	<code>subscribe_newsletter</code>	declarative form tool (<code>toolname</code> , <code>tooldescription</code> , <code>toolparamdescription</code> , <code>toolautosubmit</code>); answers via <code>SubmitEvent.respondWith()</code> ; double opt-in
knot page	<code>redeem_this_knot</code>	starts the redemption; the wallet asks the user to approve
offer page	<code>redeem_this_offer</code>	same, with an optional end-to-end encrypted note for the giver
purchase page	<code>decrypt_this_knot</code>	decrypts locally; keys never leave the device
purchase page	<code>pass_this_knot_on</code>	starts a pass-on (3.7); the user confirms and signs
give page	<code>fill_give_form</code>	fills title, pitch, description, editions and reward; the user adds files and publishes

A.3 Discovery and REST

<code>/llms.txt</code>	site map for language models [11]: what Giver.EU is, rules for agents, tools, pages, documents
<code>/api/openapi.json</code>	OpenAPI description of the REST API (public reads; writes need a wallet session and return unsigned transactions)
<code>/robots.txt</code>	allows <code>/api/mcp</code> and <code>/api/openapi.json</code> , points to <code>/llms.txt</code>
JSON-LD	schema.org Product on knot pages (<code>/k/<id></code>), FAQPage, VideoObject
Licences	GET <code>/api/license/<knot>/key/<key></code> , GET <code>/api/license/<knot>/wallet/<wallet></code>

Security model. Agent interfaces add no new authority: the MCP server is read-only, browser tools act with the user's session in the user's tab, and every value-moving step still requires a signature in the user's wallet. Tool descriptions instruct agents to act only on the user's explicit request.