udrive: Permissionless, Verifiable File Storage with a Consent-Based Data Market on Solana

udrive Labs

June 2026 · Version 1.0

Abstract

We present udrive, a file storage service that lets anyone publish a file and receive a durable, shareable link without registering an account, and lets holders of a Solana keypair claim a private drive by signing a message rather than by trusting a password. udrive separates three concerns that are usually conflated: who may read a file, who attests to its origin, and who consents to its later use. Read access is governed by an explicit visibility setting and, for private data, by server side or end to end encryption. Origin is established by an optional detached signature over a canonical statement that binds the file name, its content hash, and its size to a public key, so that any third party can verify authorship without contacting the uploader. Consent to downstream use is expressed through an opt in data market in which an owner offers a file for model training at a stated price and accrues rewards as usage is recorded. We give the authentication protocol, the storage path, the layered encryption model, the provenance scheme, the sharing model, and the economic design in precise terms, and we state the trust assumptions and current limitations plainly, including that on chain settlement of rewards is not yet enabled in the present release.

Keywords: wallet authentication, ed25519, authenticated encryption, content addressing, capability security, data markets, Solana.

1. Introduction

File sharing on the public internet has converged on a small number of custodial platforms. These platforms are convenient, but they couple three rights that need not be coupled. The right to read a file, the right to claim that one produced it, and the right to decide whether it may be used to train a model are, in practice, all delegated to the operator of the platform. The operator decides who may sign up, which files may remain online, and whether stored data is fed into other systems. The user is left to trust that these decisions match their intent.

udrive is an attempt to decouple those rights and to return each of them to the person who uploads a file. The system is permissionless in the sense that publishing requires no account, and self custodial in the sense that a private drive is controlled by a cryptographic key rather than by a credential held on a server. The central claim of this paper is modest and concrete. With a small amount of well understood cryptography, namely ed25519 signatures over canonical messages and authenticated symmetric encryption, a storage service can offer anonymous publishing, key based identity, verifiable provenance, and a consent based market for data, while keeping its trust assumptions small and legible.

The remainder of this paper proceeds as follows. Section 2 states the design goals and the matching non goals. Section 3 gives the architecture and the lifecycle of a request. Sections 4 through 8 describe authentication, the storage path, the encryption model, provenance, and

sharing, each with the relevant construction made explicit. Section 9 describes the data market and Section 10 its economics. Sections 11 and 12 give the security model and the measures that resist abuse. Section 13 lists the present limitations without euphemism. Sections 14 and 15 cover related work and the roadmap, and Section 16 concludes.

2. Design Goals and Non-Goals

The design is organized around five goals. First, publishing must be possible without an account, because a requirement to register is itself a gate. Second, identity for the private drive must rest on a key the user already controls, so that there is no password to leak and no recovery flow to subvert. Third, provenance must be verifiable by anyone, offline, from the file and its metadata alone. Fourth, the conditions under which data may be reused must be set by the owner and recorded, not inferred. Fifth, the service must degrade safely. When a dependency such as the rate limit store is unavailable, the system should deny sensitive operations rather than wave them through.

The non goals are equally important. udrive does not try to be a consensus system. It does not place file bytes on chain, and it does not require a token to read or write. It does not promise anonymity at the network layer, since an uploader who wishes to remain unlinkable must still take care at the transport level. It does not claim that public files are private. A file marked public is treated as public, and the only protection it receives is that the storage bucket is never browsable and that object access is mediated by short lived signed URLs.

3. System Architecture

3.1 *Components*

udrive is a web application with a small set of server side services. An application server renders pages and exposes a JSON API. A relational database stores file records, users, share links, and the bookkeeping for the data market. An in memory store holds short lived state such as authentication nonces and rate limit counters. An object store holds the file bytes. The object store is addressed only through signed URLs with brief lifetimes, and its bucket is configured to reject public listing.

A design rule that recurs throughout the system is that the application server never proxies file bytes. Uploads go directly from the browser to the object store using a signed `PUT` URL, and downloads are a redirect to a signed `GET` URL. This keeps the application server stateless with respect to payload size and removes a class of resource exhaustion attacks, at the cost of requiring the completion step described in Section 5 to confirm that the promised bytes actually arrived.

3.2 *Lifecycle of a request*

Every state changing request passes through a single guard layer before it reaches business logic. The guard resolves the client address from a platform set header rather than from a client supplied forwarding header, enforces that the request originates from an allowed origin, optionally requires a valid session, and applies a sliding window rate limit keyed by route and identity. Read paths that are sensitive, such as a private download, additionally check ownership. The guard

converts any thrown application error into a generic response so that internal details are never returned to the caller.

4. Authentication: Sign-In with Solana

A user proves control of a wallet by signing a structured message. The flow has two round trips. The client first requests a nonce for a stated public key pk. The server generates a single use nonce n, stores it with a short expiry bound to that key, and returns a canonical message that embeds the domain, a uniform resource identifier, a version, the nonce, and an issuance time t_0. The canonical message has the fixed form

```
udrive wants you to sign in with your Solana account:
<base58(pk)>

Sign in to udrive. This request will not trigger a
blockchain transaction or cost any gas fees.

URI: https://udrive.app
Version: 1
Nonce: <n>
Issued At: <t0>
```

Write $M(\mathrm{pk}, n, t_0)$ for the function that deterministically produces these bytes. The client signs the exact message with the wallet, yielding a detached ed25519 signature $\sigma = \mathrm{Sign}_{\mathrm{sk}}\big(M(\mathrm{pk}, n, t_0)\big)$, and returns $(\mathrm{pk}, n, t_0, \sigma)$.

The verification step is deliberately strict. The server does not trust any message string sent by the client. It reconstructs the canonical message as $M' = M(\mathrm{pk}, n, t_0)$ using its own fixed domain and identifier, and accepts the login only if the signature verifies against the reconstruction and the nonce is still live:

$$\mathrm{Verify}\colon \{0,1\}^{*} \times \{0,1\}^{256} \times \{0,1\}^{512} \longrightarrow \{0,1\} \tag{1}$$

$$\mathrm{accept} \Leftrightarrow \mathrm{Verify}\big(\mathrm{pk}, M', \sigma\big) = 1 \;\wedge\; \mathrm{fresh}\big(n\big) \tag{2}$$

Here Verify is the ed25519 verification predicate, which for public key point A, base point B, and signature (R, S) returns 1 exactly when the cofactored group equation holds:

$$8SB = 8R + 8H(R \,\|\, A \,\|\, M)\,A \tag{3}$$

where H is SHA-512. The nonce is consumed atomically before any signature check, so that $\mathrm{fresh}(n)$ is true for at most one verification and a captured signature cannot be replayed even under concurrent attempts. Inputs that are not a well formed thirty two byte key and sixty four byte signature are rejected before verification. On success the server issues a session token with a pinned algorithm and a bounded lifetime, delivered in a cookie marked HTTP only, restricted to same site use, and marked secure in production. Because the domain is embedded in M, a signature collected for one origin cannot be presented at another.

5. The Storage Path

5.1 *Presigned object transfer*

To upload, the client sends the file name, size, and declared content type, along with the chosen visibility and any provenance or encryption options. The server validates these against per tier limits, reserves a file record in a pending state, derives an object key from a random identifier so that keys are unguessable and cannot collide or overwrite, and returns a signed `PUT` URL with a lifetime measured in tens of seconds. The browser uploads the bytes directly. The presigned download URL is similarly short lived and always carries a content disposition of attachment unless the file is of a type that is known to be safe to display inline.

Admission is gated by a storage quota. Let u_{used} be the bytes already accounted to an owner, $s = \text{size}(F)$ the size of the incoming file F, and q_u the owner quota for their tier. The reservation is admitted only if

$$u_{\text{used}} + \text{size}\left(F\right) \le q_u \tag{4}$$

5.2 *Integrity and content addressing*

Because the application server does not see the bytes, it cannot take the uploader at their word that the object exists or matches the declared size. The completion step closes this gap. After the direct upload, the client calls a completion endpoint, and the server issues a head request to the object store to confirm that the object is present and that its size matches the reservation. Only then does the file record move from pending to active and the owner accounting $u_{\text{used}} \leftarrow u_{\text{used}} + s$ update. The client also computes a content hash in the browser,

$$h = \text{SHA-256}\left(F\right), \quad h \in \{0,1\}^{256} \tag{5}$$

and submits it, which gives a stable content identifier for display and for the provenance statement described next. Collision resistance of SHA-256 means that two distinct files are overwhelmingly unlikely to share a hash, so h can stand in for the content in later checks.

6. Visibility and the Layered Encryption Model

Every file carries a visibility and an encryption mode, and the two are related but distinct. Visibility is one of public, unlisted, or private. A public file is reachable by its link and may be surfaced by the owner. An unlisted file is reachable only by those who hold its link. A private file is reachable only by its owner. Anonymous uploads may be public or unlisted but never private, because there is no key to bind them to.

Encryption is layered so that protection is proportional to intent. Public and unlisted files are stored without encryption, since their contents are meant to be served on request, but they are still placed in a private bucket and served only through signed URLs. Private files default to server side encryption with a service held key $k \in \{0,1\}^{256}$ using AES-256 in Galois/Counter Mode. With a fresh random initialization vector iv, encryption of plaintext m produces a ciphertext and an authentication tag,

$$\left(c, t\right) = \text{Enc}_k\left(\text{iv}, m\right), \quad \text{Dec}_k\left(\text{iv}, c, t\right) = \begin{cases} m & \text{if tag valid} \\ \bot & \text{otherwise} \end{cases} \tag{6}$$

so that a leak of object storage alone reveals neither contents nor an undetectable forgery: any tampering with c or t makes decryption return the failure symbol $\perp$. For the strongest setting, a signed in user may choose end to end encryption, in which the browser derives a key from a wallet signature over a fixed label and encrypts the file before it leaves the device. In that mode the server stores only ciphertext and a small metadata record, and it cannot decrypt the file. The cost of this mode is that previews are unavailable and that loss of the wallet means loss of the data, which is the expected trade for custody.

7. Provenance: Wallet-Attested Authorship

Provenance answers a different question from access. It does not ask who may read a file but who is willing to put their name to it. An uploader may attach a detached signature over a canonical provenance statement. The statement is a short, fixed format text that names the file, its content hash h, its size s, the signer public key pk, and a timestamp t. Writing $P = \mathrm{Stmt}\big(\mathrm{name}, h, s, \mathrm{pk}, t\big)$ for the statement, the uploader supplies $\sigma_P = \mathrm{Sign}_{\mathrm{sk}}\big(P\big)$, and the server stores both alongside the file record.

Verification is public and repeatable. A verification endpoint recomputes the verdict from stored fields, accepting only if the signature is valid over the stored statement and the statement still describes the file as stored:

$$\mathrm{valid} \;\Leftrightarrow\; \mathrm{Verify}\big(\mathrm{pk}, P, \sigma_P\big) = 1 \;\;\wedge\;\; h = \mathrm{SHA\text{-}256}\big(F\big) \tag{7}$$

The second conjunct means that swapping the bytes under a previously valid proof is detected and reported as a mismatch rather than silently accepted. Because the statement binds the content hash, a verifier learns not only that a key signed something but that the key signed this exact content. The scheme makes no claim about legal authorship. It states a cryptographic fact, namely that the holder of a given key attested to a given file at a given time.

8. Capability-Based Sharing

Sharing is modeled as an unforgeable capability rather than as an access control list. An owner mints a share link, which is a record carrying a random token τ and optional constraints: a password, an expiry, and a maximum number of downloads. The token is the capability. Possession of it, together with the password if one is set, authorizes download up to the stated limit. Passwords are never stored in the clear. A password pw is reduced to a salted scrypt hash,

$$\mathrm{dk} = \mathrm{scrypt}\big(\mathrm{pw}, \mathrm{salt}, N, r, p\big) \tag{8}$$

where N is the CPU and memory cost, r the block size, and p the parallelization factor; the memory hardness of scrypt raises the cost of offline guessing. A presented password is accepted only if its derived key equals the stored dk under the stored salt. The link can be revoked at any time, after which it fails closed. This model keeps sharing decentralized in spirit, since the owner can hand out a capability without the recipient needing an account, while preserving the owner ability to bound and withdraw access.

9. A Consent-Based Data Market

9.1 *Opt-in and pricing*

The data market is strictly opt in. A file is eligible for model training only if its owner enables training on that file and sets a price $\text{price}(f) \geq 0$. Nothing is enrolled by default, and anonymous uploads are never enrolled. A price of zero leaves a file out of the market entirely. The intent is to invert the usual arrangement, in which data is collected first and consent is sought later or not at all. Here consent is a precondition recorded against the specific file.

9.2 *Accrual and settlement*

When a model consumes an enrolled file, the system records a training event e that references the file f_e, the consuming model, and a measure of usage, and it accrues a reward to the owner at the file stated price. Let $\mathcal{E}_u$ be the set of events that reference a file owned by user u. The accrued balance is the sum of the prices of the files consumed,

$$A_u = \sum_{e \in \mathcal{E}_u} \text{price}\left(f_e\right) \tag{9}$$

and the amount the owner may withdraw is the accrued total net of what has already been claimed. Writing L_u for the cumulative claimed balance,

$$C_u^{\text{claimable}} = A_u - L_u \geq 0 \tag{10}$$

The owner sees a running balance of accrued and claimable rewards and a history of events. We disclose plainly that, in the present release, the settlement of these rewards is not performed on chain. Balances accrue and the claim action sets $L_u \leftarrow A_u$, moving the balance to a settled state for accounting, but no transfer of value is broadcast to any network. Section 13 restates this limitation, and Section 15 describes the path to real settlement.

10. Incentive and Economic Design

The economic design rests on a simple alignment. Storage that is never read imposes cost without benefit, while data that is useful to train a model has a value that the owner is best placed to price. By letting the owner set a per file price and by metering usage, the market gives owners a reason to publish high quality data under clear terms and gives consumers a predictable cost. Pricing is per file rather than per account, which lets an owner treat different files differently, charging for a curated dataset while leaving a personal document at a price of zero and therefore out of the market. The unit of accounting is a decimal quantity carried with sufficient precision to avoid rounding drift, and the sum in Equation (10) is evaluated in that representation rather than in floating point, so that repeated accrual does not accumulate error.

11. Security Model

11.1 *Trust assumptions*

The user trusts their wallet and their own device. For public and unlisted files, and for server side encrypted private files, the user trusts the operator not to misuse access to stored bytes, exactly

as with any custodial service. For end to end encrypted files, that trust is removed, since the operator holds only ciphertext. The user trusts the object store to honor signed URLs and bucket policy, and trusts the transport layer for confidentiality in transit.

11.2 *Threat model and mitigations*

We consider an attacker who can make arbitrary requests, read public responses, and attempt to forge identity, escalate access, or exhaust resources. Identity forgery is addressed by signature verification over a server reconstructed message with single use nonces and a pinned token algorithm, as in Equations (2) and (3). Cross site request forgery is addressed by requiring a trusted origin on every state changing request and by same site cookies. Insecure direct object reference is addressed by checking ownership on private reads and on every mutation, and by returning a not found result rather than a forbidden result so that the existence of a record is not disclosed. Resource exhaustion is addressed by direct to storage transfer, by the quota inequality of Equation (4) enforced after a head check, and by the sliding window rate limit of Section 12 that fails closed for sensitive routes when the counter store is unavailable. Content smuggling, in which a file of an active type is served so that it executes in the origin, is addressed by serving downloads as attachments and by permitting inline display only for a short allow list of types, never for markup or script.

12. Abuse Resistance

Open publishing invites misuse, so the system includes lightweight controls that do not require accounts. The core control is a sliding window rate limit. For an identity and route key k, a window length W, and the set of request timestamps $R(k)$ recorded for that key, the count over the trailing window ending at time t is

$$n(k,t) = \left|\{\, r \in R(k) : t - W < r \le t \,\}\right| \tag{11}$$

and a request is admitted only while $n(k,t) < L$ for the route limit L. Anonymous uploads are additionally capped per address per day. Any file can be reported, and reports are recorded with a hashed reporter address rather than a raw one. When the count of open reports for a file crosses a threshold, the file is quarantined automatically, after which its view and download paths return a not found result until a human review occurs. Addresses are never stored in the clear. They are reduced to keyed, non reversible fingerprints using a dedicated derived key, so that rate limiting and abuse accounting do not create a log of who downloaded what.

13. Limitations and Honest Disclosures

We state the current limits directly. First, reward settlement is not on chain in this release. The market records consent, usage, and accrual, and the user interface presents the feature as active, but no value is transferred on any network; the claim action is an accounting only stub that advances L_u to A_u. Second, storage is custodial. The default deployment uses a managed object store, so the strongest privacy is available only through the end to end encrypted mode. Third, the system does not provide network level anonymity. Fourth, automatic quarantine is a blunt instrument and can be triggered by coordinated false reports, which is why quarantine suspends

rather than deletes and is reversible by review. We prefer to publish these limits rather than to imply guarantees the system does not yet make.

14. Related Work

udrive draws on several established lines. Sign in with Ethereum [1] and the broader effort to standardize wallet based authentication motivate the message format and the emphasis on domain binding and single use nonces. The signature scheme itself is ed25519 [3]. Content addressed and decentralized storage systems, including the interplanetary file system [2] and contract based storage networks, motivate the use of a content hash as a stable identifier and inform the roadmap toward pluggable storage backends. Capability based security [6], with its long history in operating systems research, motivates the share link model, and the password hashing follows the memory hard design of scrypt [7]. Rate limiting as a defense against resource abuse echoes the pricing via processing idea [4]. The market design borrows from the literature on data dignity and on compensating contributors of training data [5], which argues that the producers of data should share in the value it creates. The settlement layer targeted by the roadmap is Solana [8].

15. Roadmap

Three directions follow naturally from the present design. The first is on chain settlement, in which accrued rewards are paid through a program on Solana and the claim action becomes a real transfer with an on chain record, replacing the present accounting only stub. The second is pluggable storage, in which the object store behind the signed URL abstraction is replaced or complemented by a decentralized backend, so that durability no longer rests on a single operator. The third is verifiable usage, in which a model consuming an enrolled file can produce a succinct proof that a particular file was included in a training run, so that accrual is backed by evidence rather than by a trusted report. Each of these strengthens a specific trust assumption identified in Section 11.

16. Conclusion

udrive shows that anonymous publishing, key based identity, verifiable provenance, and consent based reuse can coexist in one service without a large trusted base. The construction is deliberately conventional in its cryptography and careful in its defaults, failing closed where safety and availability conflict and disclosing where the present system stops short of its goals. We view the current release as a foundation. The work that remains, settlement, decentralized durability, and proofs of usage, is the work of making each promise in this paper not only stated but enforced.

References

1. Sign-In with Ethereum (EIP-4361). Specification for off-chain authentication using account signatures, 2021.
2. Benet, J. IPFS: Content Addressed, Versioned, Peer-to-Peer File System, 2014.
3. Bernstein, D. J., Duif, N., Lange, T., Schwabe, P., Yang, B. High-speed high-security signatures (Ed25519), 2012.
4. Dwork, C., Naor, M. Pricing via Processing, or Combatting Junk Mail, 1992.

5. Lanier, J., Weyl, E. G. Data as Labor. In Radical Markets, 2018.
6. Miller, M. S. Robust Composition: Towards a Unified Approach to Access Control and Concurrency Control, 2006.
7. Percival, C. Stronger Key Derivation via Sequential Memory-Hard Functions (scrypt), 2009.
8. Yakovenko, A. Solana: A New Architecture for a High Performance Blockchain, 2017.

udrive Labs. This document describes the system as implemented at Version 1.0. Where the running service and this paper differ, the honest disclosures in Section 13 take precedence.