BLOCKCHAIN COMMONS

HUBERT

GORDIAN DEV MEETING

WHAT IS HUBERT?

- A protocol that facilites *secure multiparty transactions:*
  - 📝 **Participants write once** using random keys
  - 🆔 **Messages contain random keys** for expected responses, enabling bidirectional communication
  - 📦 **Complete opacity** to outsiders through end-to-end encryption
  - 🛜 **No central server** required for coordination
  - 🤝 **Trustless operation** using public distributed networks

WHAT ARE URS?

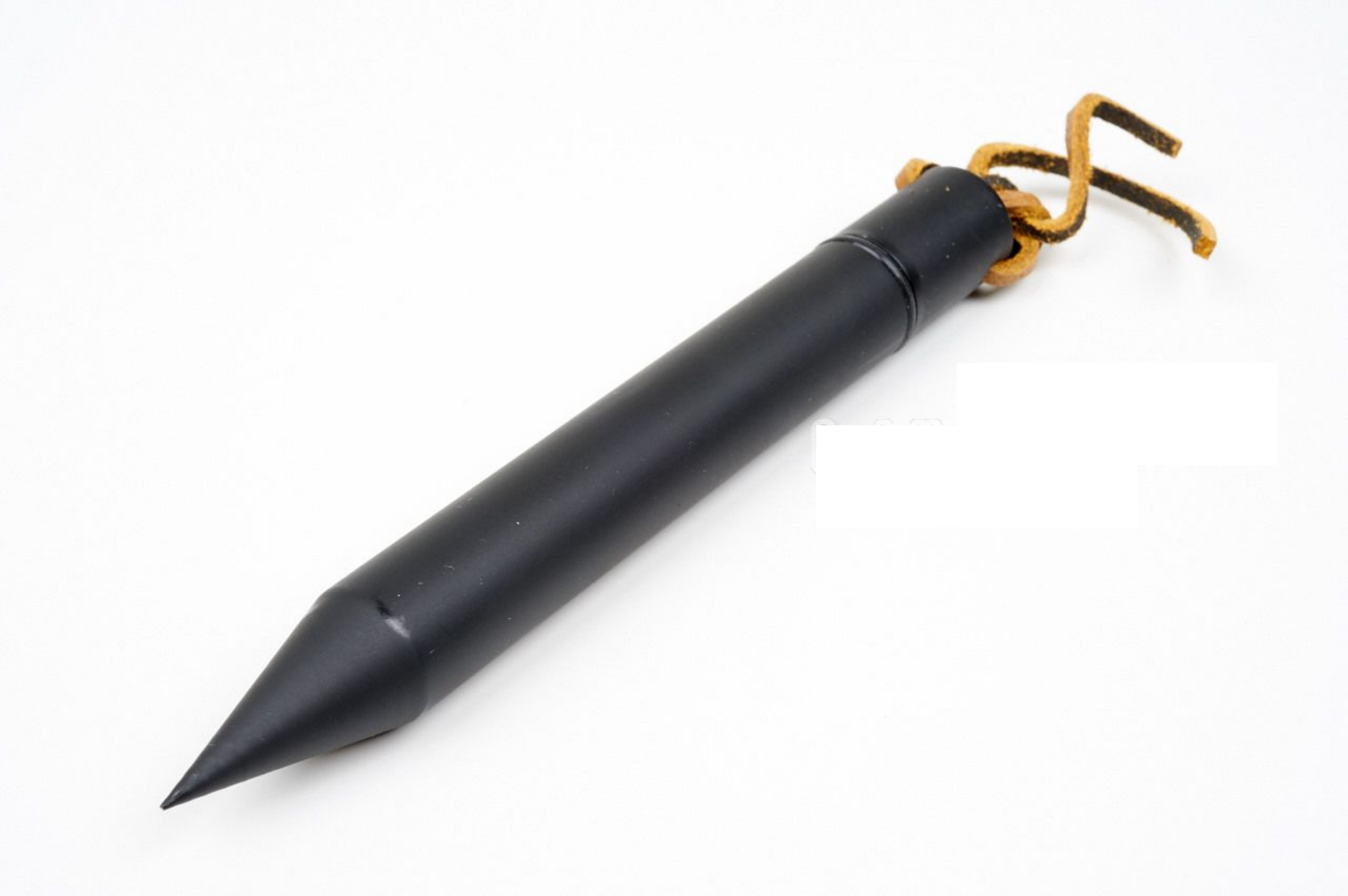
- UR: Uniform Resource
  - Defined in **BCR-2020-005**
    - https://github.com/blockchaincommons/research
  - Encodes binary data as typed, easy to handle text URI
  - ur:type/bytewords

WHAT IS AN ARID?

- 🆔 ARID: Apparently Random Identifier
  - Defined in **BCR-2022-002**
    - https://github.com/blockchaincommons/research
  - 256 statistically random bits (32 bytes)
  - Can refer to anything
  - But cannot be correlated to anything
  - In Hubert, ARIDs are addresses of cryptographic *dead drops.*

WHAT IS A DEAD DROP?

- Spycraft: A way to exchange messages where the parties never have to meet in person.
- Sender leaves message at known location, receiver visits later and picks it up.
- Requires out-of-band coordination to set up locations, protocols, and signals.



WHAT IS GORDIAN ENVELOPE?

- "Smart document" format for structured data with a built-in Merkle-like digest tree.
- Core superpower: hashed-data elision (selective redaction) with verifiable inclusion proofs–disclose only what's needed while signatures remain valid.
- Radically recursive: envelopes contain other envelopes.
- Privacy and human-rights oriented design (RFC 6973 / RFC 8280 alignment).

GORDIAN ENVELOPE

GORDIAN DEV MEETING

WHAT IS GORDIAN ENVELOPE?

- Flexible cryptography–Schnorr, Ed25519, SSH, Post-Quantum algorithms supported for signing/encryption.
- Supports non-correlation via salt.
- Suited to cryptographic seeds, keys, verifiable credentials, and other high-integrity artifacts
- Compact binary format with minimal overhead.
- Wire-agnostic: integrity, privacy, and semantics travel with the data–ideal for offline/air-gapped and multi-hop workflows.

GORDIAN ENVELOPE

GORDIAN DEV MEETING

WHAT IS GORDIAN SEALED TRANSACTION PROTOCOL (GSTP)?

- Secure, transport-agnostic request/response protocol built on Gordian Envelope
- Works over HTTP/TCP, Bluetooth, NFC, QR “sneakernet,” Tor, etc.
- Messages are both encrypted to recipients and signed by the sender
- Provides Encrypted State Continuations (ESC) to carry encrypted workflow state inside the messages themselves.

WHAT IS GORDIAN SEALED TRANSACTION PROTOCOL (GSTP)?

- Asynchronous and peer-to-peer friendly; also supports classic client-server flows and multimodal (multi-channel) transactions.
- Encourages idempotent actions so retries over flaky links are safe.
- Designed to be “wire-agnostic”: encryption/signing are at the protocol layer, so insecure/unreliable transports are acceptable.

STORAGE LAYERS

HUBERT

BitTorrent™

IPFS

SQLite

GORDIAN DEV MEETING

STORAGE LAYERS

BitTorrent™

GORDIAN DEV
MEETING

STORAGE: BITTORRENT MAINLINE DHT

BitTorrent™

- Used by BitTorrent clients to find peers.
- In 2013, the concurrent number of users of Mainline DHT was 16-28 million, with intra-day changes of at least 10 million.
- UDP overlay where peers collectively replace trackers; keys live in a 160-bit space and lookups walk “closest” nodes.
- “Sloppy” DHT—eventually consistent and best-effort. Items expire without re-announcement.
- Signaling/coordination layer, not durable storage.

GORDIAN DEV MEETING

STORAGE: BITTORRENT MAINLINE DHT

BitTorrent™

- Two data modes:
  - Immutable items (key = SHA-1 of the value)
  - Mutable items (key = SHA-1 of the publisher's Ed25519 public key concatenated with optional salt)
- Values are practically capped at ~1 KB.
- Expect messages to last ~30–45 minutes without republishing

GORDIAN DEV MEETING

STORAGE: BITTORRENT MAINLINE DHT

BitTorrent™

- We use the “mutable” mode by stretching the ARID into an Ed25519 key pair.
- We derive the Ed25519 private key from the ARID deterministically using HKDF:
  - Private key used for signing the payload
  - Public key derived from private key, used to compute DHT storage location and verify payload
  - Write-once semantics enforced by rejecting updates past seq=1

GORDIAN DEV MEETING

STORAGE: IPFS

IPFS

- IPFS (InterPlanetary File System) is a distributed, content-addressed storage network
- Established network with consistent operation; well-suited for coordination and signaling. ~56,000 dedicated infrastructure nodes globally
- Content persists reliably when pinned; 48-hour DHT expiration ensures fresh routing data
- Content addressing: Files identified by CID (Content Identifier) = cryptographic hash of content

GORDIAN DEV MEETING

STORAGE LAYERS

IPFS

GORDIAN DEV MEETING

STORAGE: IPFS

IPFS

- Two-layer storage model:
  - Immutable layer: Data stored at CID (content hash), retrieved via Kademlia DHT
  - Mutable layer: IPNS (InterPlanetary Name System) provides mutable pointers to CIDs
- Size limits: Practical limit ~1-10 MB per item (much larger than DHT's 1 KB)
- Persistence: Content expires from DHT in 48 hours without re-announcement
  - IPNS records should be republished every 4 hours (Kubo default)
  - Long-term persistence requires "pinning" (explicit storage on specific nodes)
- Latency: 1-10 seconds for IPNS resolution (slower than Mainline DHT)

GORDIAN DEV MEETING

STORAGE: IPFS

IPFS

- Dependencies: Requires Kubo daemon or compatible IPFS node with RPC API
- Write-once semantics: Before publishing, check if IPNS name resolves to any CID
  - If resolution succeeds → name already published → reject with AlreadyExists error
  - If resolution fails with "could not resolve name" → name unpublished → proceed with publish
  - Unlike Mainline DHT's sequence number approach, relies on IPNS name existence check

GORDIAN DEV MEETING

STORAGE LAYERS

BitTorrent™

IPFS

GORDIAN DEV MEETING

STORAGE: HYBRID

BitTorrent IPFS

- Hybrid storage combines Mainline DHT and IPFS with automatic size-based routing
- Intelligent storage selection:
  - Small envelopes (≤1000 bytes): Stored directly in Mainline DHT
  - Large envelopes (>1000 bytes): Two-step indirection process
- Storage process for large envelopes:
  - 1. Generate new "reference ARID" for the actual envelope
  - 2. Store actual envelope in IPFS at reference ARID
  - 3. Store small reference envelope in DHT at original ARID

GORDIAN DEV MEETING

STORAGE: HYBRID

- Retrieval process:
  - 1. Fetch envelope from DHT using original ARID
  - 2. Check if it's a reference envelope (has 'dereferenceVia': "ipfs")
  - 3. If reference: Extract reference ARID and fetch actual envelope from IPFS
  - 4. If not reference: Return DHT envelope directly

STORAGE: HYBRID

BitTorrent IPFS

- Benefits:
  - Transparent to caller: Application uses same API regardless of storage backend
  - Fast retrieval for small messages (DHT: 1-5 seconds)
  - Large capacity for big messages (IPFS: up to 10 MB)
  - No size calculation required by caller

GORDIAN DEV MEETING

STORAGE: HYBRID

BitTorrent IPFS

- Dependencies: Requires both embedded DHT client and Kubo daemon for IPFS
- Write-once semantics: Enforced separately by each backend
  - (DHT seq=1, IPFS existence check)
- Still to be implemented:
  - Encryption of reference envelope to multiple recipients; currently stored as plaintext and exposes IPFS ARID.

GORDIAN DEV MEETING

STORAGE LAYERS

SQLite

GORDIAN DEV MEETING

STORAGE: SERVER

- Centralized HTTP-based storage for testing and controlled deployments
- Architecture: Simple HTTP POST API with write-once semantics
  - PUT endpoint: /put (stores ARID + envelope + optional TTL)
  - GET endpoint: /get (retrieves envelope by ARID with polling)
- Two storage backends:
  - Memory: In-memory HashMap, volatile (lost on restart)
  - SQLite: Persistent file-based database storage

STORAGE: SERVER

- Storage features:
    - Direct ARID-to-envelope mapping (no key derivation like DHT/IPFS)
    - Write-once enforcement: Returns 409 CONFLICT if ARID already exists
    - TTL support: Optional time-to-live in seconds for automatic expiration
    - Background cleanup: Expired entries pruned automatically (every 60 seconds for SQLite)


DEMO
HUBERT