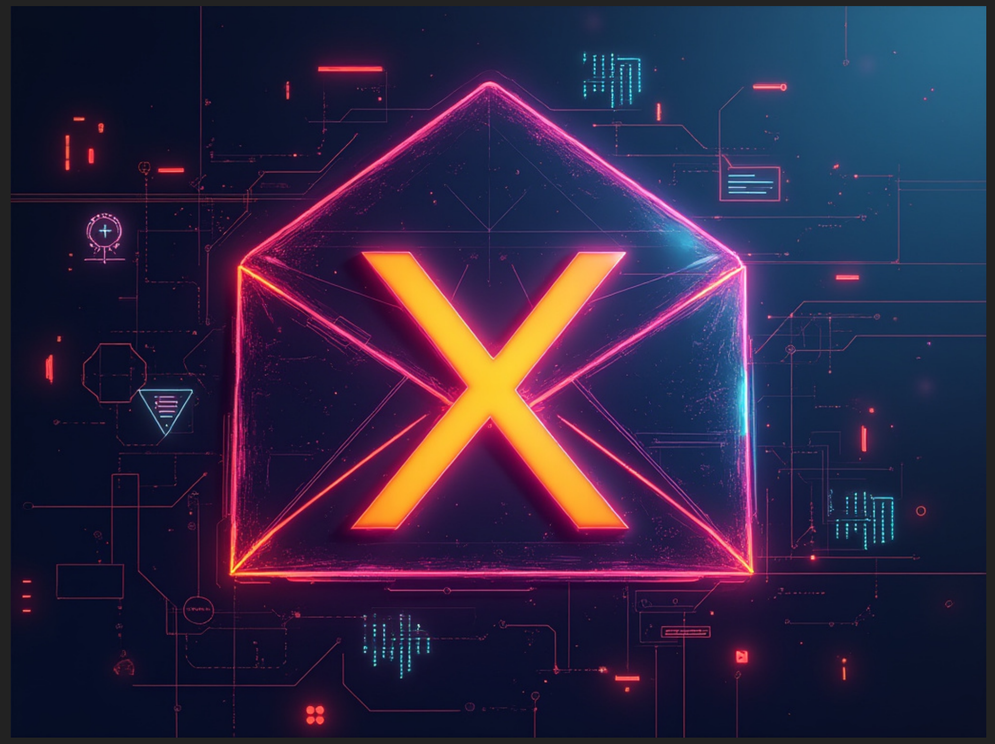
BLOCKCHAIN COMMONS

XID: EXTENSIBLE IDENTIFIERS

WHAT IS BLOCKCHAIN COMMONS?

- We are a community that brings together stakeholders to collaboratively build open & interoperable, secure & compassionate infrastructure.
- We design decentralized solutions where everyone wins.
- We are a neutral “not-for-profit” that enables people to control their own digital destiny.



INTRODUCING XID

INTRODUCING XID

- eXtensible IDentifier (say *"zid"*)
- A decentralized identifier
- Uses Gordian Envelope to go beyond the capabilities of W3C DIDs, DID Controller Documents, and W3C Verifiable Credentials
- Early research document at HackMD.io– please comment!
  - Early Rust implementation in progress
  - No formal specification
  - No test vectors
    - ...yet.
- What does a XID look like?

INTRODUCING XID

XID(71274df133169a0e2d2ffb11cbc7917732acafa31989f685cca6cb69d473b93c)

- 32-byte unique identifier with a XID CBOR tag

INTRODUCING XID

```
40024(
   h'71274df133169a0e2d2ffb11cbc7917732acafa31989f685cca6cb69d473b93c'
)

d9 9c58                                 # tag(40024) xid
   5820                                 # bytes(32)
      71274df133169a0e2d2ffb11cbc7917732acafa31989f685cca6cb69d473b93c
```

- The CBOR structure adds only 5 bytes.

INTRODUCING XID

XID(71274df1)

- For users, we'll usually abbreviate to the first four bytes
- These can be considered the XID's human-readable "name"
- So we want ways to make this name as recognizable as possible

ⓧ INTRODUCING XID

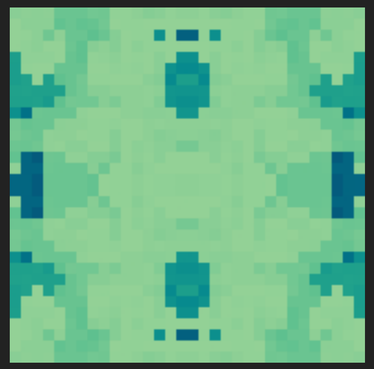
XID(71274df1)

ⓧ JUGS DELI GIFT WHEN

ⓧ🌊😹🌽🏓

- The “name” can be translated to Bytewords for easy recognition
- We're also experimenting with using *Bytemoji*

INTRODUCING XID



XID(71274df1)

JUGS
DELI
GIFT
WHEN

- Or for even quicker recognition, present a unique LifeHash
- Bytewords and Bytemoji use only 4 bytes of the XID (unique enough)
- But LifeHash uses all 32 bytes (extremely unique!)
- These techniques can be combined

INTRODUCING XID

XID(71274df1)

ur:xid/hdcxjsdigtwneocmnybadpdlzobysbstmekteypspeotcfldynlpsfolsbintyjkrhfnvsbyrdfw

- A XID can be converted to a URI using the Uniform Resource (UR) format.
- How do we access the information a XID points to?

RESOLVING A XID

RESOLVING A XID

XID(71274df1)

- A XID by itself doesn’t include a resolution method.
- If there is a universal resolver method available, then this is all you need.

RESOLVING A XID

```
XID(71274df1) [
    'dereferenceVia': "https://resolver.example.com"
    'dereferenceVia': "btcr:01234567"
]
```

- But if we want custom methods, we can compose a little Gordian Envelope and add as many as we want!
- Resolving a XID returns a *XID Document*.
- In fact, a XID with resolution methods *is* a tiny XID document!

RESOLVING A XID

```
XID(2d9296d0) [
    'dereferenceVia': "https://resolver.example.com"
    'dereferenceVia': "btcr:01234567"
]
```

ur:xid:lfaehddpswmdtphgaeaewkjsmojezojtwfbaoxkgndpybyayamftahjnmyeobyehecrhothhpffd
esueskaahpvwrtincxueftrdemeyihtngdgmgwhflfae

- This envelope can also be encoded as a ur:xid
- It’s longer, but looks and acts just the same as a bare XID.
- So any ur:xid can be completely *self-describing*.

CREATING A XID

CREATING A XID

PrivateSigningKey/PublicSigningKey

CBOR(PublicSigningKey)

SHA256(CBOR(PublicSigningKey))

71274df133169a0e2d2ffb11cbc7917732acafa31989f685cca6cb69d473b93c

- Generate a key pair (Ed25519, BIP-340 Schnorr, ECDSA...)
  - We recommend BIP-340 due to its ease of use with MuSig2 and FROST
  - This is called the *genesis key*.
- Encode it to CBOR (Unique to the key and type)
- Take the SHA-256 digest of the CBOR
- That's your XID!

CREATING A XID

XID(71274df133169a0e2d2ffb11cbc7917732acafa31989f685cca6cb69d473b93c)

- Showing that the XID was produced from a public key proves:
  - Global uniqueness (because the private key is a unique secret)
  - Randomness (because SHA-256 is used on the public key)
- XIDs are guaranteed to be uniformly random sequences of numbers
- So how do you present these proofs? Through a *XID Document*.

THE XID DOCUMENT

```
{
    XID(71274df1) [
        'allow': 'All'
        'key': PublicSigningKey
    ]
} [
    'verifiedBy': Signature
]
```

- The *XID Document* is a Gordian Envelope that packages:
  - The XID itself
  - The genesis key
  - A signature made by the genesis key
  - NOTE: The genesis key can be rotated *without* invalidating the XID!
- So what sorts of additional assertions can a XID document hold?

THE XID DOCUMENT

```
{
    XID(71274df1) [
        'allow': 'All'
        'dereferenceVia': "https://resolver.example.com"
        'key': PublicSigningKey
    ]
}[
    'verifiedBy': Signature
]
```

- Resolution methods
  - Affords discovery of the most recent revisions of the document
  - A universal resolver method could resolve bare XIDs.

THE XID DOCUMENT

```
{
    XID(71274df1) [
        'allow': 'All'
        'dereferenceVia': "https://resolver.example.com"
        'key': PublicKeyBase
    ]
}[
    'verifiedBy': Signature
]
```

- Keys for signing *and* encryption
  - Add individual keys or use a PublicKeyBase, which packages a SigningPublicKey (for signature verification) and an AgreementPublicKey (for encryption).

THE XID DOCUMENT

```
{
    XID(71274df1) [
        'allow': 'all'
        'controller': XID(28ab67d4)
        'dereferenceVia': "https://resolver.example.com"
        'key': PublicKeyBase      // not the genesis key
    ]
}[
    'verifiedBy': Signature
]
```

- Permission delegation
  - XID documents use a “deny-first” permission structure
  - In this example, all permissions have been delegated away from the genesis key!
  - Resolvers and the verifiable history of a XID prove it’s still valid

THE XID DOCUMENT

```
XID(71274df1) [
    'allow': 'all'
    'dereferenceVia': "https://resolver.example.com"
    'key': PublicKeyBase
    'key': 'group' [
        'controller': XID(5802b4ff) [
            'dereferenceVia': "btc:01234567"
        ]
        'key': PublicKeyBase [
            'deny': 'verify'
            'endpoint': "https://messaging.example.com"
        ]
    ]
]
```

- Keys with fine-grained permissions
- Other resolution methods
- Endpoints for specific functionality

THE XID DOCUMENT

```
XID(71274df1) [
    'allow': 'all'
    'dereferenceVia': "https://resolver.example.com"
    'key': PublicKeyBase
    'key': 'group' [
        'controller': XID(5802b4ff) [
            'dereferenceVia': "btc:01234567"
        ]
        'key': PublicKeyBase [
            'deny': 'verify'
            'endpoint': "https://messaging.example.com"
        ]
    ]
    'provenance': ProvenanceMark      // revision 5
]
```

ProveMark.com

- Provenance Marks afford a distributed, traceable hash chain of revisions

ELIDING A XID DOCUMENT

ELIDING A XID DOCUMENT

```
{
    XID(71274df1) [
        'allow': 'all'
        'dereferenceVia': "https://resolver.example.com"
        'key': PublicKeyBase
        'key': 'group' [
            'controller': XID(5802b4ff) [
                'dereferenceVia': "btc:01234567"
            ]
            'key': PublicKeyBase [
                'deny': 'verify'
                'endpoint': "https://messaging.example.com"
            ]
        ]
        'provenance': ProvenanceMark
    ]
} [
    'verifiedBy': Signature
]
```

- XID documents can be vended with all information intact

ELIDING A XID DOCUMENT

```
{
    XID(71274df1) [
        'allow': 'all'
        'key': PublicKeyBase
        'key': 'group' [
            ELIDED
            'key': PublicKeyBase [
                'deny': 'verify'
                'endpoint': "https://messaging.example.com"
            ]
        ]
        ELIDED (2)
    ]
} [
    'verifiedBy': Signature
]
```

- Or can be elided for a specific purpose by the vendor
- And the signature still verifies!

ELIDING A XID DOCUMENT

```
{
    XID(71274df1) [
        'allow': 'all'
        'key': PublicKeyBase
        ELIDED (3)
    ]
} [
    'verifiedBy': Signature
]
```

- In fact, you can elide most of the assertions in the document, and simply reveal that you have a signed XID, and that it the enclosed key verifies it.
- You can reveal anything else it contains later as needed. This is *progressive trust*.

WHAT ARE THE USES OF XIDS?

- A new form of decentralized identifier
  - Goes beyond capabilities of W3C DID Controller Documents
  - Elision determines who gets which keys, endpoints, assertions, endorsements, and other sensitive details
- Create many new types of secure credentials & entitlements
- Many other use cases:
  - NOSTR – rotate keys, share reputation signals, and more!
  - Record provenance of digital assets (without a blockchain!)

CHRISTOPHER ALLEN
christophera@lifewithalacrity.com
@BlockchainComns

WOLF MCNALLY
wolf@wolfmcnally.com
@WolfMcNally