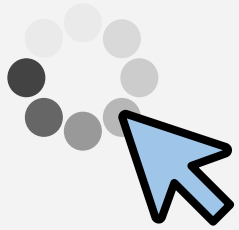
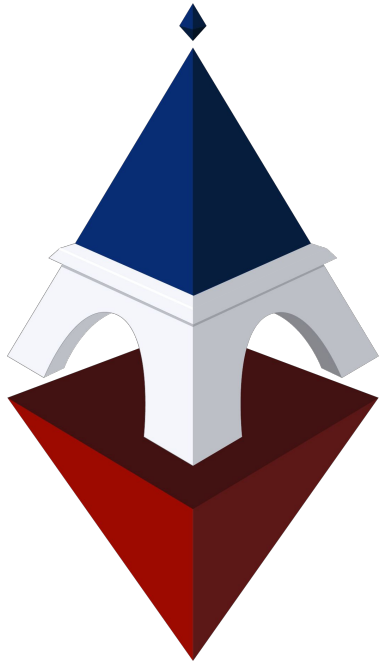
Ethereum Attestation Service



Leonid Beder - EthCC 2021



Who are we?

- **Dino Mark**: Ethereum contributor
- **Steve Dakh:** Developer and early Ethereum team member (Kryptokit/Jaxx, Rushwallet)
- **Leonid Beder:** Developer, researcher, and DeFi enthusiast (currently, Bancor Core)

Dino Mark

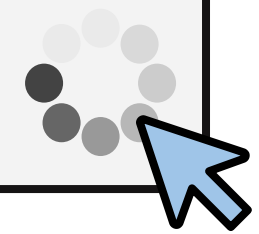
Steve Dakh

Leonid Beder

ethereum

The Problem

- On-chain trust, identity, and reputation is **necessary**, but is **extremely hard**
- Tailor-made solutions do exist, but create **fragmentation**, aren't **composable**, somewhat **expensive**, and overburden L1

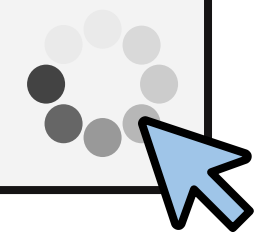


What is EAS?

- ➔ **Base layer** protocol for **global**, **generic** attestations
- ➔ Built **exclusively** on Ethereum
- ➔ **Fully open sourced**
- ➔ Fully tested
- ➔ **Permissionless**, **community owned**, and **tokenless**
- ➔ EAS requires ETH. Go buy it!
- ➔ Launching **Q4 2021**

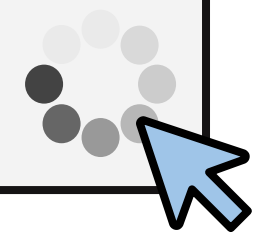
Some Use Cases

- ➔ Identity verification
- ➔ Voting
- ➔ Ticketing
- ➔ Proof of Existence
- ➔ On-chain KYC access-based permissions



Some Use Cases

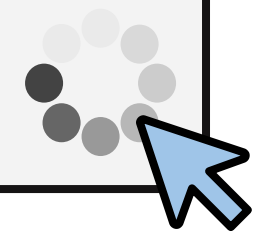
- ➔ Human clout reputation
- ➔ Trade reputation
- ➔ Credit score
- ➔ Micro-lending based on EAS reputation/Credit scores (e.g., for uncollateralized loans)
- ➔ Reputation explorers
- ➔ ... and many more



Resources

- **Contracts:** https://github.com/ethereum-attestation-service/contracts
- **SDK:** https://github.com/ethereum-attestation-service/sdk
- **Sample App:** https://github.com/ethereum-attestation-service/app

Any feedback, criticism, and contribution are more than welcome!



Design and Architecture

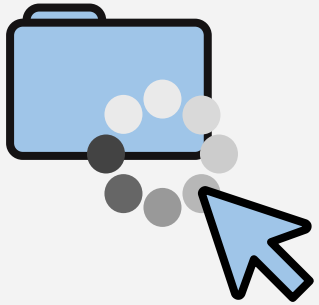
EAS.sol

Main contract where attestations are made

AORegistry.sol

Registry of all Attestation Object Schemas

Simple design based on two main contracts

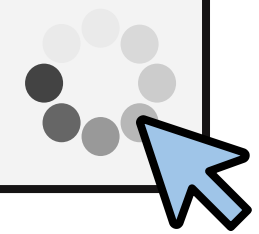


TL;DR

- ➔ Define an **Attestation Object (AO)** with a schema and an optional verifier
- ➔ Attest to any **AO** directly or delegate via an **EIP712** typed signature message
- ➔ Revoke your attestations directly or delegate via an **EIP712** typed signature message
- ➔ Link related attestations using their UUIDs

Dev Stack

- Solidity 0.7.6
- ABI coder v2
- Hardhat
- ethers
- Waffle
- TypeScript




<<Interface>>
IAOVerifier
External:
verify(recipient: address, schema: bytes, data: bytes, expirationTime: uint256, msgSender: address, msgValue: uint256): bool
<<Interface>>
IAORegistry
External:
register(schema: bytes, verifier: IAOVerifier): bytes32
getAO(uuid: bytes32): AORecord
getAOCount(): uint256
Public:
<<event>> Registered(uuid: bytes32, index: uint256, schema: bytes, verifier: IAOVerifier, attester: address)
<<Interface>>
IEIP712Verifier
External:
getNonce(account: address): uint256
attest(recipient: address, ao: bytes32, expirationTime: uint256, refUUID: bytes32, data: bytes, attester: address, v: uint8, r: bytes32, s: bytes32)
revoke(uuid: bytes32, attester: address, v: uint8, r: bytes32, s: bytes32)
<<Interface>>
IEAS
External:
getAORegistry(): IAORegistry
getEIP712Verifier(): IEIP712Verifier
getAttestationsCount(): uint256
attest(recipient: address, ao: bytes32, expirationTime: uint256, refUUID: bytes32, data: bytes): bytes32
attestByDelegation(recipient: address, ao: bytes32, expirationTime: uint256, refUUID: bytes32, data: bytes, attester: address, v: uint8, r: bytes32, s: bytes32): bytes32
revoke(uuid: bytes32)
revokeByDelegation(uuid: bytes32, attester: address, v: uint8, r: bytes32, s: bytes32)
getAttestation(uuid: bytes32): Attestation
isAttestationValid(uuid: bytes32): bool
getReceivedAttestationUUIDs(recipient: address, ao: bytes32, start: uint256, length: uint256, reverseOrder: bool): bytes32[]
getReceivedAttestationUUIDsCount(recipient: address, ao: bytes32): uint256
getSentAttestationUUIDs(attester: address, ao: bytes32, start: uint256, length: uint256, reverseOrder: bool): bytes32[]
getSentAttestationUUIDsCount(recipient: address, ao: bytes32): uint256
getRelatedAttestationUUIDs(uuid: bytes32, start: uint256, length: uint256, reverseOrder: bool): bytes32[]
getRelatedAttestationUUIDsCount(uuid: bytes32): uint256
Public:
<<event>> Attested(recipient: address, attester: address, uuid: bytes32, ao: bytes32)
<<event>> Revoked(recipient: address, attester: address, uuid: bytes32, ao: bytes32)
EIP712Verifier
External:
getNonce(account: address): uint256
attest(recipient: address, ao: bytes32, expirationTime: uint256, refUUID: bytes32, data: bytes, attester: address, v: uint8, r: bytes32, s: bytes32)
revoke(uuid: bytes32, attester: address, v: uint8, r: bytes32, s: bytes32)
Public:
constructor()
AORegistry
Private:
_getUUID(ao: AORecord): bytes32
External:
register(schema: bytes, verifier: IAOVerifier): bytes32
getAO(uuid: bytes32): AORecord
getAOCount(): uint256
EAS
Private:
_attest(recipient: address, ao: bytes32, expirationTime: uint256, refUUID: bytes32, data: bytes, attester: address): bytes32
_revoke(uuid: bytes32, attester: address)
_getUUID(attestation: Attestation): bytes32
_sliceUUIDs(uuids: bytes32[], start: uint256, length: uint256, reverseOrder: bool): bytes32[]
External:
getAORegistry(): IAORegistry
getEIP712Verifier(): IEIP712Verifier
getAttestationsCount(): uint256
getAttestation(uuid: bytes32): Attestation
getReceivedAttestationUUIDs(recipient: address, ao: bytes32, start: uint256, length: uint256, reverseOrder: bool): bytes32[]
getReceivedAttestationUUIDsCount(recipient: address, ao: bytes32): uint256
getSentAttestationUUIDs(attester: address, ao: bytes32, start: uint256, length: uint256, reverseOrder: bool): bytes32[]
getSentAttestationUUIDsCount(recipient: address, ao: bytes32): uint256
getRelatedAttestationUUIDs(uuid: bytes32, start: uint256, length: uint256, reverseOrder: bool): bytes32[]
getRelatedAttestationUUIDsCount(uuid: bytes32): uint256
Public:
<<payable>> attest(recipient: address, ao: bytes32, expirationTime: uint256, refUUID: bytes32, data: bytes): bytes32
<<payable>> attestByDelegation(recipient: address, ao: bytes32, expirationTime: uint256, refUUID: bytes32, data: bytes, attester: address, v: uint8, r: bytes32, s: bytes32): bytes32
constructor(registry: IAORegistry, verifier: IEIP712Verifier)
revoke(uuid: bytes32)
revokeByDelegation(uuid: bytes32, attester: address, v: uint8, r: bytes32, s: bytes32)
isAttestationValid(uuid: bytes32): bool

Attestation Record (AO)

- ➔ **schema:** custom specification of the attestation type, e.g. **(uint8 rating)**, **(uint256 bookISBN)**
- ➔ **verifier:** schema verifier (optional)
- ➔ **uuid:** derived from the **schema** and the **verifier**
- ➔ **index:** auto-incremented and assigned by the registry

```
/**
 * @title A struct representing a record for a submitted AO (A
 */
struct AORecord {
    // A unique identifier of the AO.
    bytes32 uuid;
    // Optional schema verifier.
    IAOVerifier verifier;
    // Auto-incrementing index for reference, assigned by the
    uint256 index;
    // Custom specification of the AO (e.g., an ABI).
    bytes schema;
}
```

Attest

A **msg.sender** attests to **recipient** on attestation **UUID** with custom **data**

The attestation expires in **expirationTime** seconds

The attestation can optional relate to **refUUID**

```
/**
 * @dev Attests to a specific AO.
 *
 * @param recipient The recipient of the attestation.
 * @param ao The UIID of the AO.
 * @param expirationTime The expiration time of the attestat
 * @param refUUID An optional related attestation's UUID.
 * @param data Additional custom data.
 *
 * @return The UUID of the new attestation.
 */
function attest(
    address recipient,
    bytes32 ao,
    uint256 expirationTime,
    bytes32 refUUID,
    bytes calldata data
) external payable returns (bytes32);
```

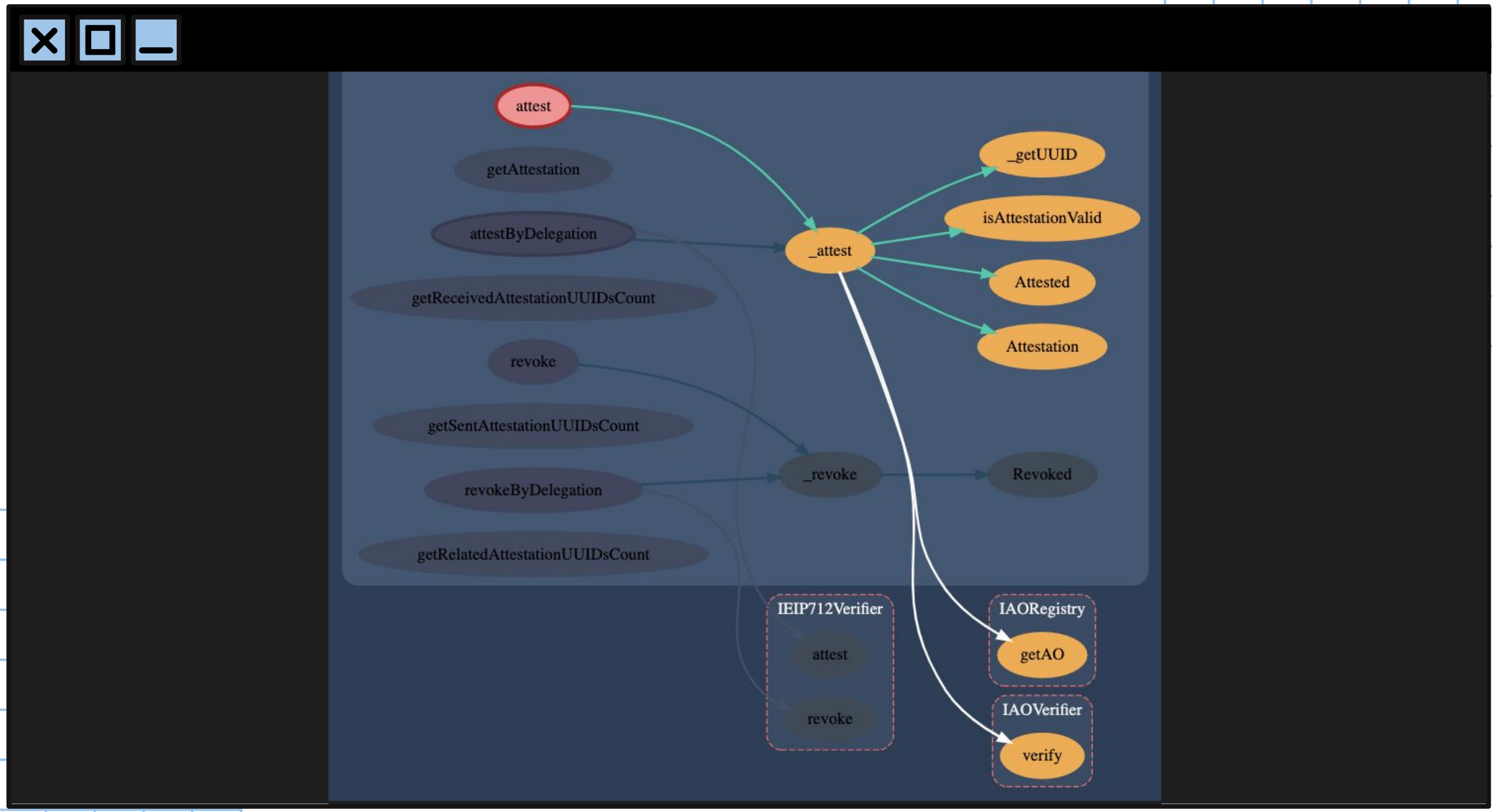
Attest

A **msg.sender** attests to **recipient** on attestation **UUID** with custom **data**

The attestation expires in **expirationTime** seconds

The attestation can optional relate to **refUUID**

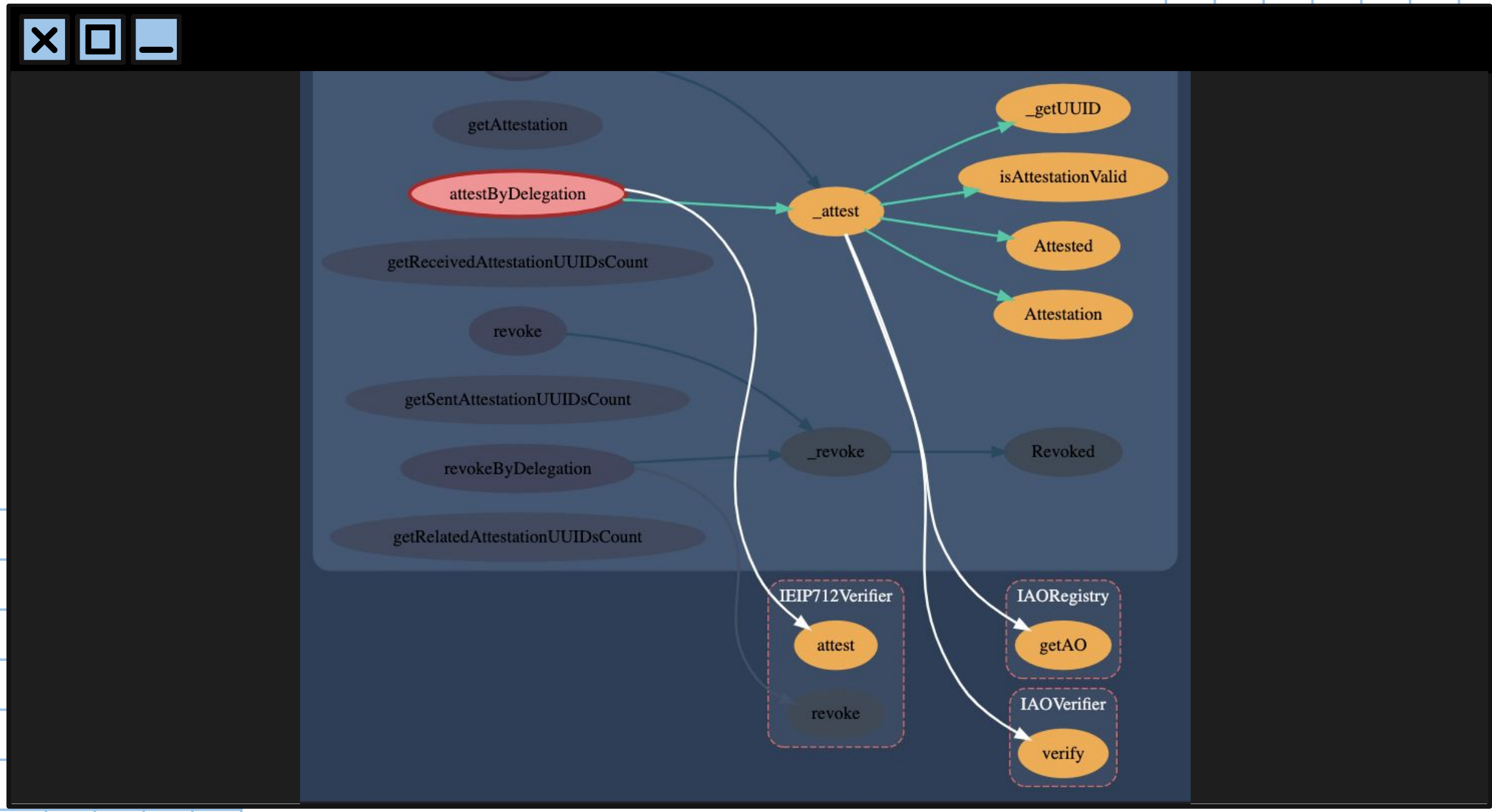
```
/**
 * @dev A struct representing a single attestation.
 */
struct Attestation {
    // A unique identifier of the attestation.
    bytes32 uuid;
    // A unique identifier of the AO.
    bytes32 ao;
    // The recipient of the attestation.
    address recipient;
    // The attester/sender of the attestation.
    address attester;
    // The time when the attestation was created (Unix timestamp).
    uint256 time;
    // The time when the attestation expires (Unix timestamp).
    uint256 expirationTime;
    // The time when the attestation was revoked (Unix timestamp).
    uint256 revocationTime;
    // The UUID of the related attestation.
    bytes32 refUUID;
    // Custom attestation data.
    bytes data;
}
```


attest
getAttestation
attestByDelegation
getReceivedAttestationUUIDsCount
revoke
getSentAttestationUUIDsCount
revokeByDelegation
getRelatedAttestationUUIDsCount
_attest
_revoke
_getUUID
isAttestationValid
Attested
Attestation
Revoked
IEIP712Verifier
attest
revoke
IAORegistry
getAO
IAOVerifier
verify

Attest via EIP712

An **attester** delegates her attestation request to **msg.sender** via a EIP712 typed signature on an **ATTEST_TYPEHASH** message

```
/**
 * @dev Attests to a specific AO using a provided EIP712 signature.
 *
 * @param recipient The recipient of the attestation.
 * @param ao The UUID of the AO.
 * @param expirationTime The expiration time of the attestation.
 * @param refUUID An optional related attestation's UUID.
 * @param data Additional custom data.
 * @param attester The attesting account.
 * @param v The recovery ID.
 * @param r The x-coordinate of the nonce R.
 * @param s The signature data.
 *
 * @return The UUID of the new attestation.
 */
function attestByDelegation(
    address recipient,
    bytes32 ao,
    uint256 expirationTime,
    bytes32 refUUID,
    bytes calldata data,
    address attester,
    uint8 v,
    bytes32 r,
    bytes32 s
) external payable returns (bytes32);
```


getAttestation
attestByDelegation
getReceivedAttestationUUIDsCount
revoke
getSentAttestationUUIDsCount
revokeByDelegation
getRelatedAttestationUUIDsCount
_attest
_revoke
_getUUID
isAttestationValid
Attested
Attestation
Revoked
IEIP712Verifier
attest
revoke
IAORegistry
getAO
IAOVerifier
verify

```
58      function attest(
59          address recipient,
60          bytes32 ao,
61          uint256 expirationTime,
62          bytes32 refUUID,
63          bytes calldata data,
64          address attester,
65          uint8 v,
66          bytes32 r,
67          bytes32 s
68      ) external override {
69          bytes32 digest = keccak256(
70              abi.encodePacked(
71                  "\x19\x01",
72                  DOMAIN_SEPARATOR,
73                  keccak256(abi.encode(
74                      ATTEST_TYPEHASH,
75                      recipient,
76                      ao,
77                      expirationTime,
78                      refUUID,
79                      keccak256(data),
80                      _nonces[attester]++
81                  ))
82              )
83          );
84
85          address recoveredAddress = ecrecover(digest, v, r, s);
86          require(recoveredAddress != address(0) && recoveredAddress == attester, "ERR_INVALID_SIGNATURE");
87      }
```

Using the SDK

Provide a callback to sign a **EIP712AttestationRequest** and that's it

Can be used to construct **EIP712** requests via a HW, HSM, KMS, local key, etc.

```
async getAttestationRequest(
  recipient: string | SignerWithAddress,
  ao: string,
  expirationTime: BigNumber,
  refUUID: string,
  data: string,
  nonce: BigNumber,
  privateKey: string
) {
  return this.delegation.getAttestationRequest(
    {
      recipient: typeof recipient === 'string' ? recipient : recipient.address,
      ao,
      expirationTime,
      refUUID,
      data: Buffer.from(data.slice(2), 'hex'),
      nonce
    },
    async (message) => {
      const { v, r, s } = ecsign(message, Buffer.from(privateKey, 'hex'));
      return { v, r, s };
    }
  );
}
```

Schema Verifcation

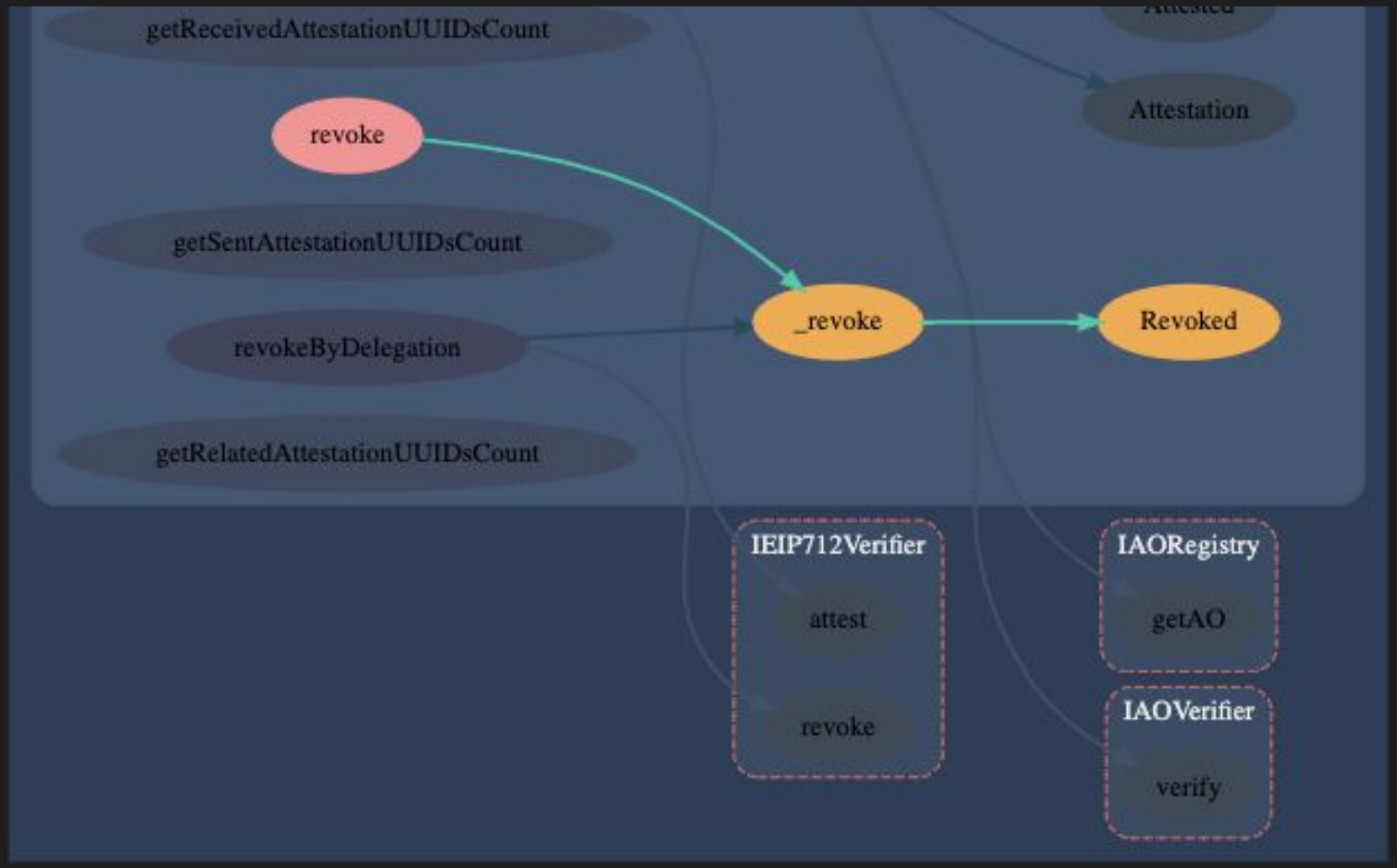
An optional custom schema on-chain verification, set by the registrar of the schema, and is invoked by the **EAS** contract

```
interface IAOVerifier {
    /**
     * @dev Verifies whether the specified attestation data conforms to the
     *
     * @param recipient The recipient of the attestation.
     * @param schema The AO data schema.
     * @param data The actual attestation data.
     * @param expirationTime The expiration time of the attestation.
     * @param msgSender The sender of the original attestation message.
     * @param msgValue The number of wei send with the original attestation
     *
     * @return Whether the data is valid according to the scheme.
     */
    function verify(
        address recipient,
        bytes calldata schema,
        bytes calldata data,
        uint256 expirationTime,
        address msgSender,
        uint256 msgValue
    ) external view returns (bool);
}
```

Revoke

A **msg.sender** revokes a previous attestation **UUID**

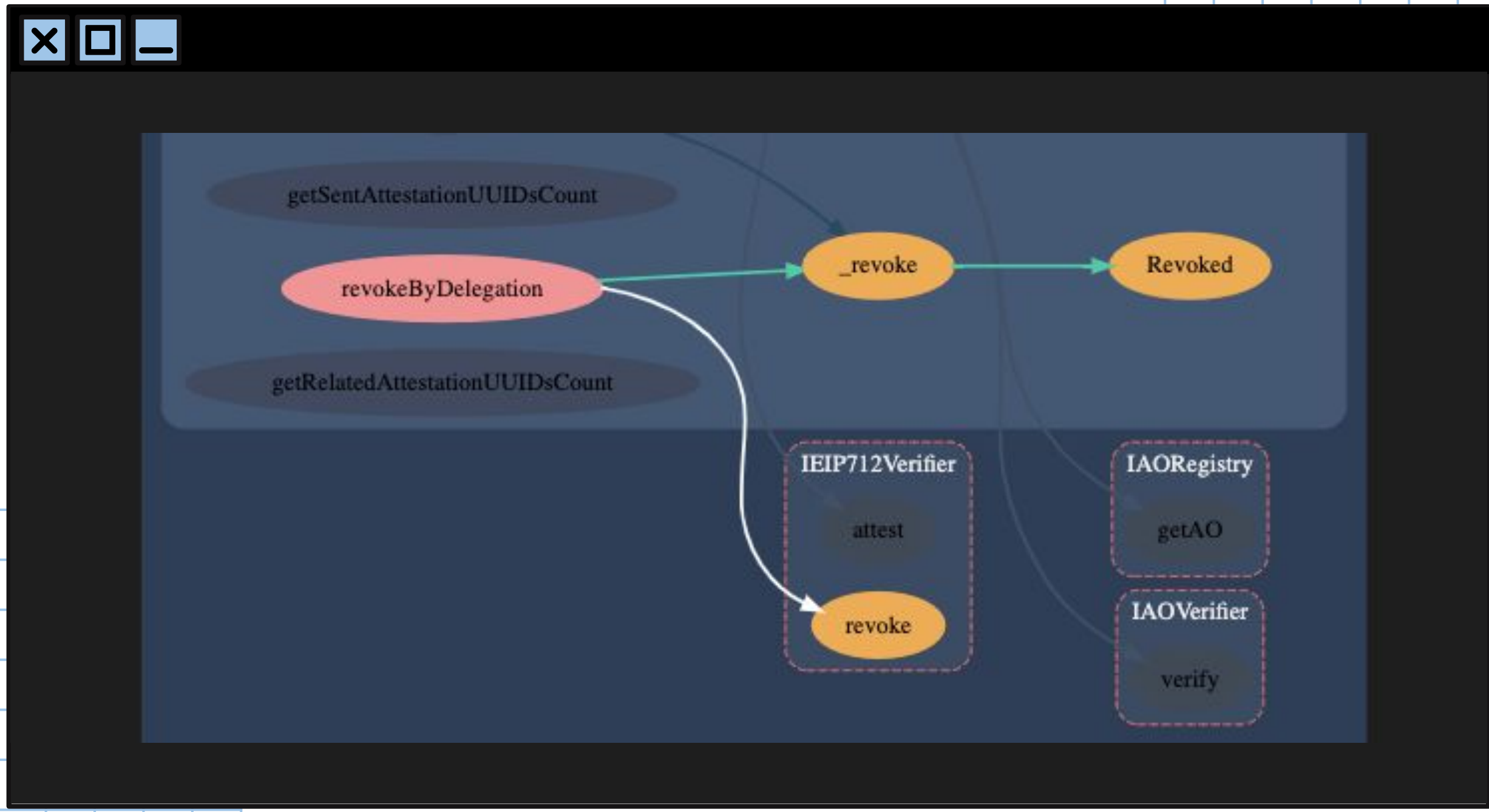
```
/**
 * @dev Revokes an existing attestation to a specific AO.
 *
 * @param uuid The UUID of the attestation to revoke.
 */
function revoke(bytes32 uuid) external;
```


getReceivedAttestationUUIDsCount
revoke
Attestation
getSentAttestationUUIDsCount
_revoke
Revoked
revokeByDelegation
getRelatedAttestationUUIDsCount
IEIP712Verifier
attest
revoke
IAORegistry
getAO
IAOVerifier
verify

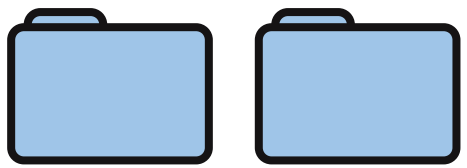
Revoke via EIP712

An **attester** delegates her revocation request to **msg.sender** via a **EIP712** typed signature on an **REVOKE_TYPEHASH** message

```
/**
 * @dev Attests to a specific AO using a provided EIP712 signa
 *
 * @param uuid The UUID of the attestation to revoke.
 * @param attester The attesting account.
 * @param v The recovery ID.
 * @param r The x-coordinate of the nonce R.
 * @param s The signature data.
 */
function revokeByDelegation(
    bytes32 uuid,
    address attester,
    uint8 v,
    bytes32 r,
    bytes32 s
) external;
```


getSentAttestationUUIDsCount
revokeByDelegation
_revoke
Revoked
getRelatedAttestationUUIDsCount
IEIP712Verifier
attest
revoke
IAORegistry
getAO
IAOVerifier
verify

Examples

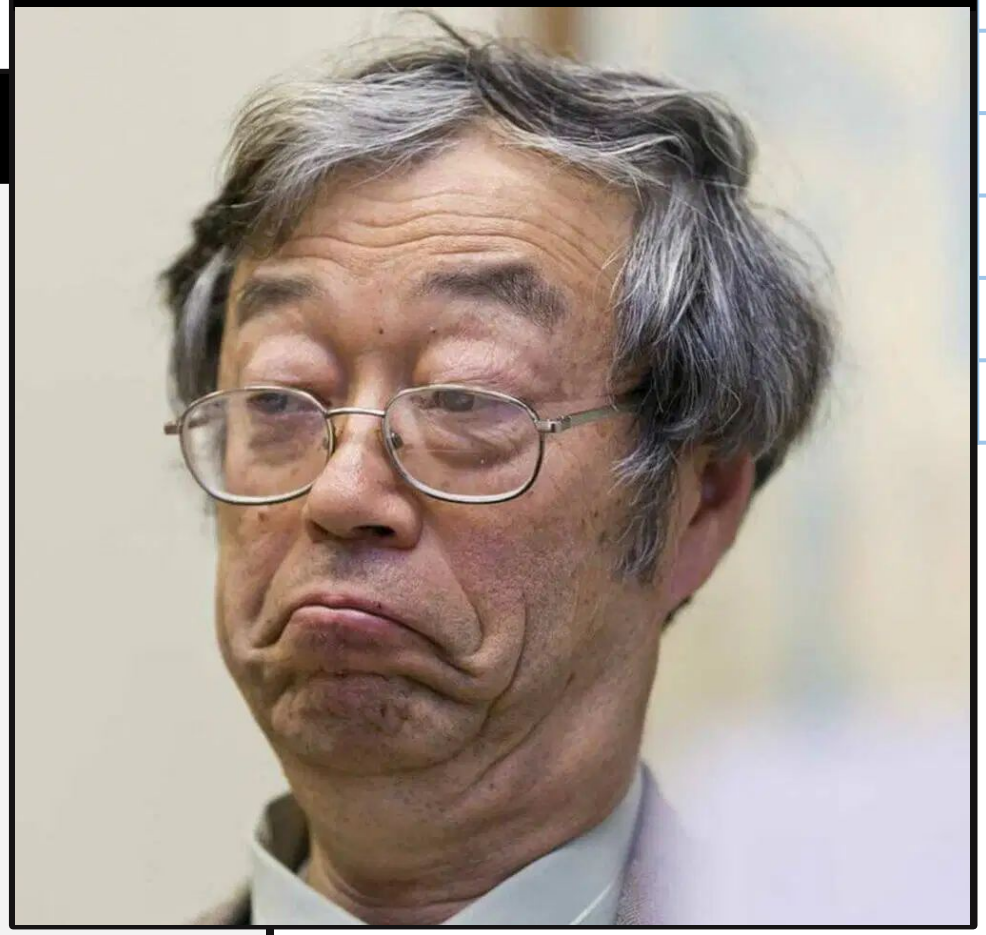


KYC

Attesting to an identity of an Ethereum account

AO Schema:

(bytes32 firstName, bytes32 lastName, bytes32 passportNumber, bool isSatoshiNakamoto)

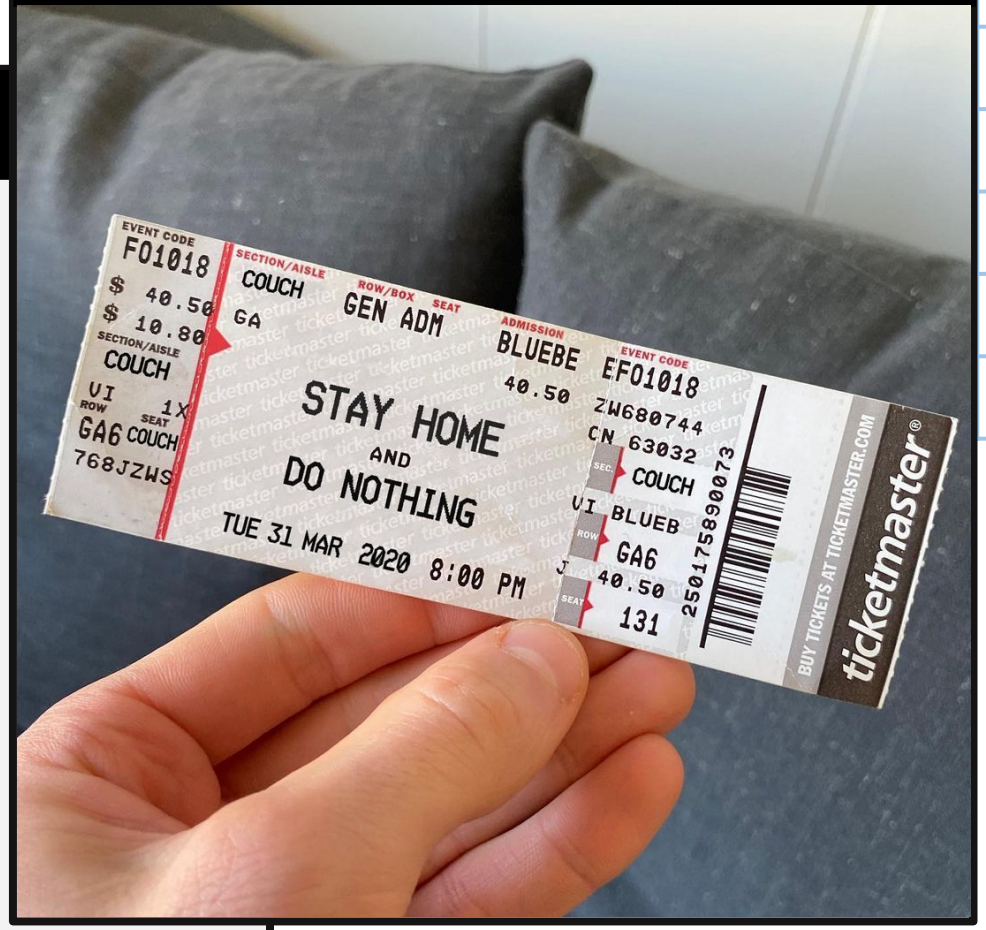


Ticketing System

Attesting to ownership of a ticket

AO Schema:

(bytes32 eventID, uint256 ticketType, uint256 seatNumber, uint256 price)

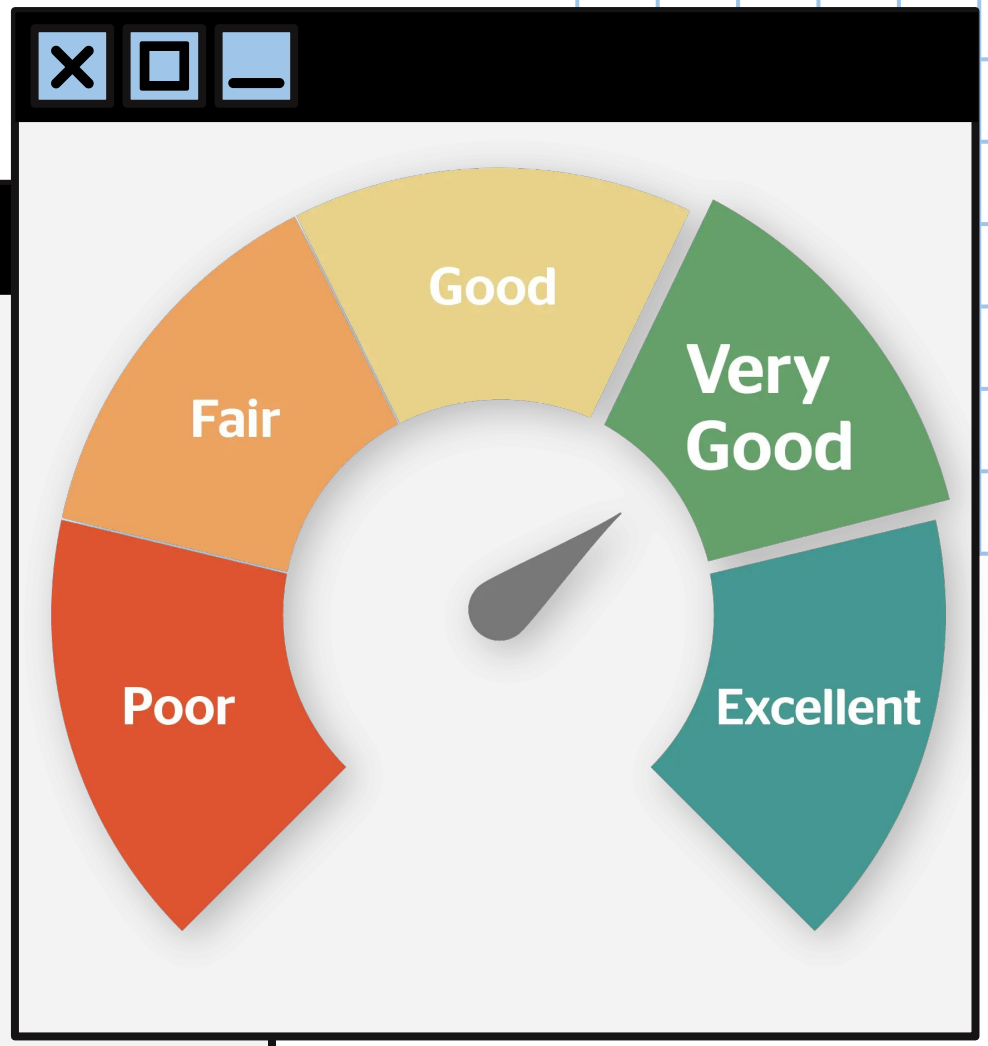


Reputation/ Clout System

Giving a score via an attestation

AO Schema:

```
(uint256 rating)
```

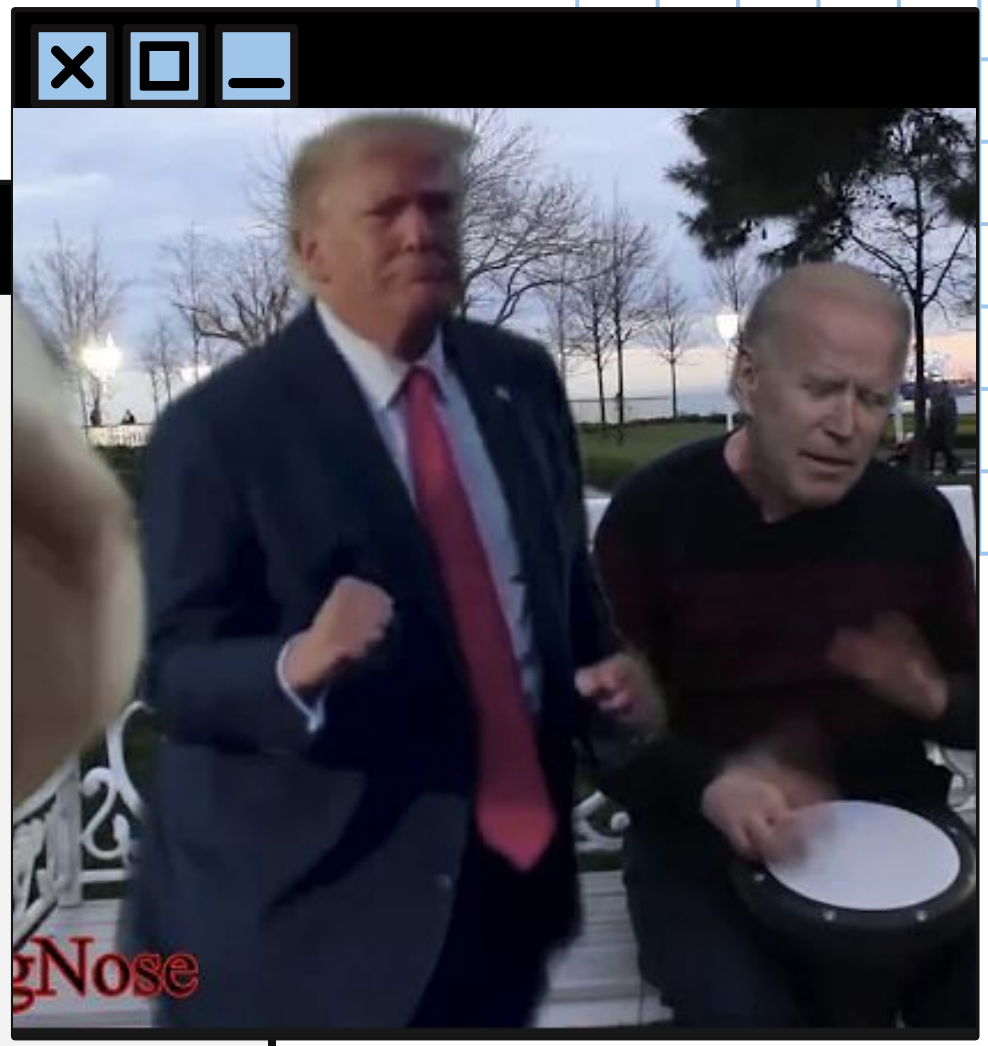


Voting

Transparent voting systems are trivial to implement. Secret ballots with zk proofs?

AO Schema:

`(bytes32 proposal, uint8 vote)`



Land Registry

A lands authority attests to an ownership of land

AO Schema:

```
(bytes polygonArea, uint8 landType,
uint8, expirty )
```

Oracles

Attest to the outcome real world events.

AO Schema:

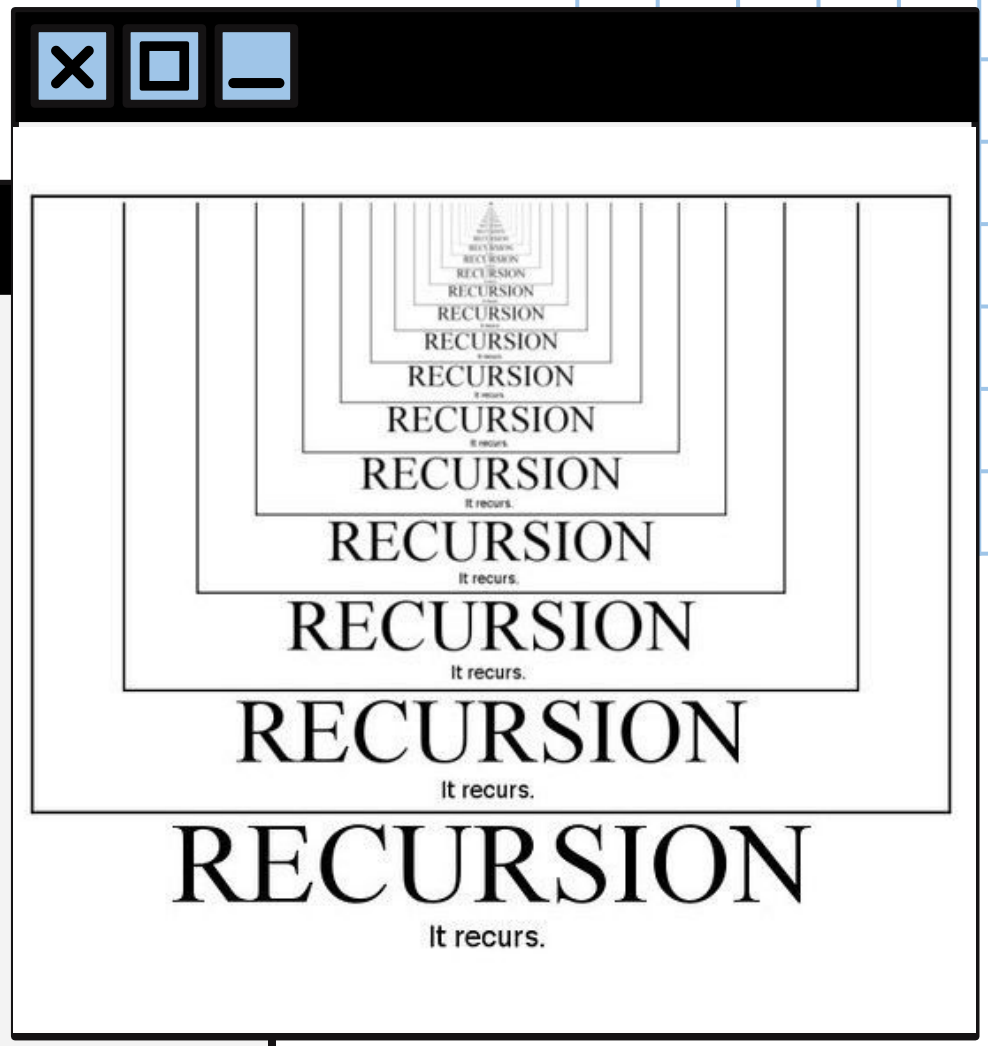
```
(bytes32 realWorldEventID, bool outcome)
```

INCEPTION

Attest to the accuracy of an attestation on another attestation about the accuracy of another attestation.

AO Schema:

```
(bool trueFalse)
```



Thank you!