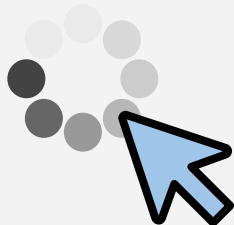
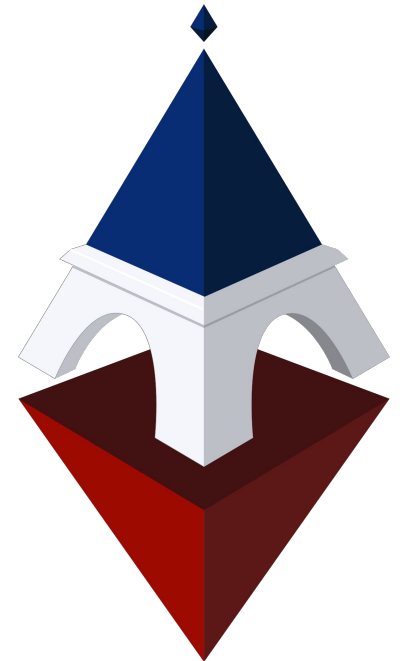


Ethereum Attestation Service



Leonid Beder - EthCC 07/22/21



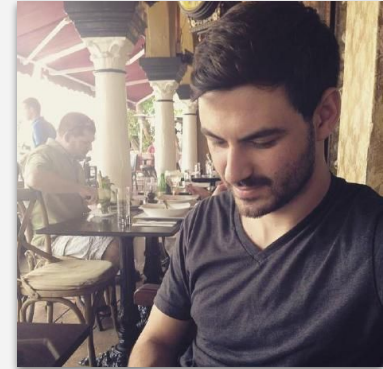


Who are we?

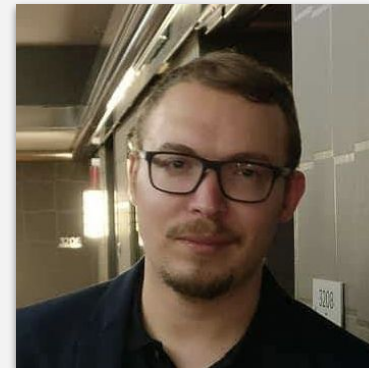
- **Dino Mark:** Ethereum contributor
- **Steve Dakh:** Developer and early Ethereum team member (Kryptokit/Jaxx, Rushwallet)
- **Leonid Beder:** Developer, researcher, and DeFi enthusiast (currently, Bancor Core)



Dino Mark



Steve Dakh



Leonid Beder

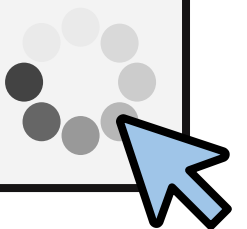


ethereum



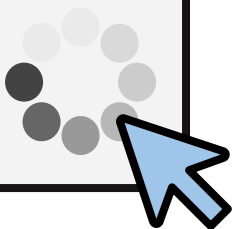
The Problem

- 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**



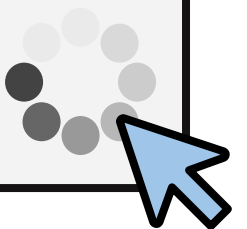


Some Use Cases

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

Some Use Cases

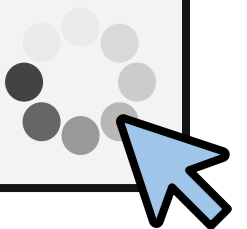
- Human clout reputation
- Trade reputation
- Credit score
- Micro-lending based on EAS reputation/Credit scores
(e.g., for uncollateralized loans)
- Reputation explorers
- ... and many more



Resources

- Website: <https://eas.eth.link/>
- Contracts: <https://github.com/ethereum-attestation-service/contracts>
- SDK: <https://github.com/ethereum-attestation-service/sdk>

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







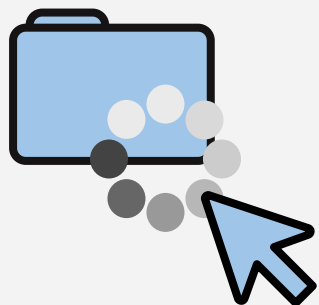
EAS.sol

Main contract
where
attestations
are made

**ASRegistry
.sol**

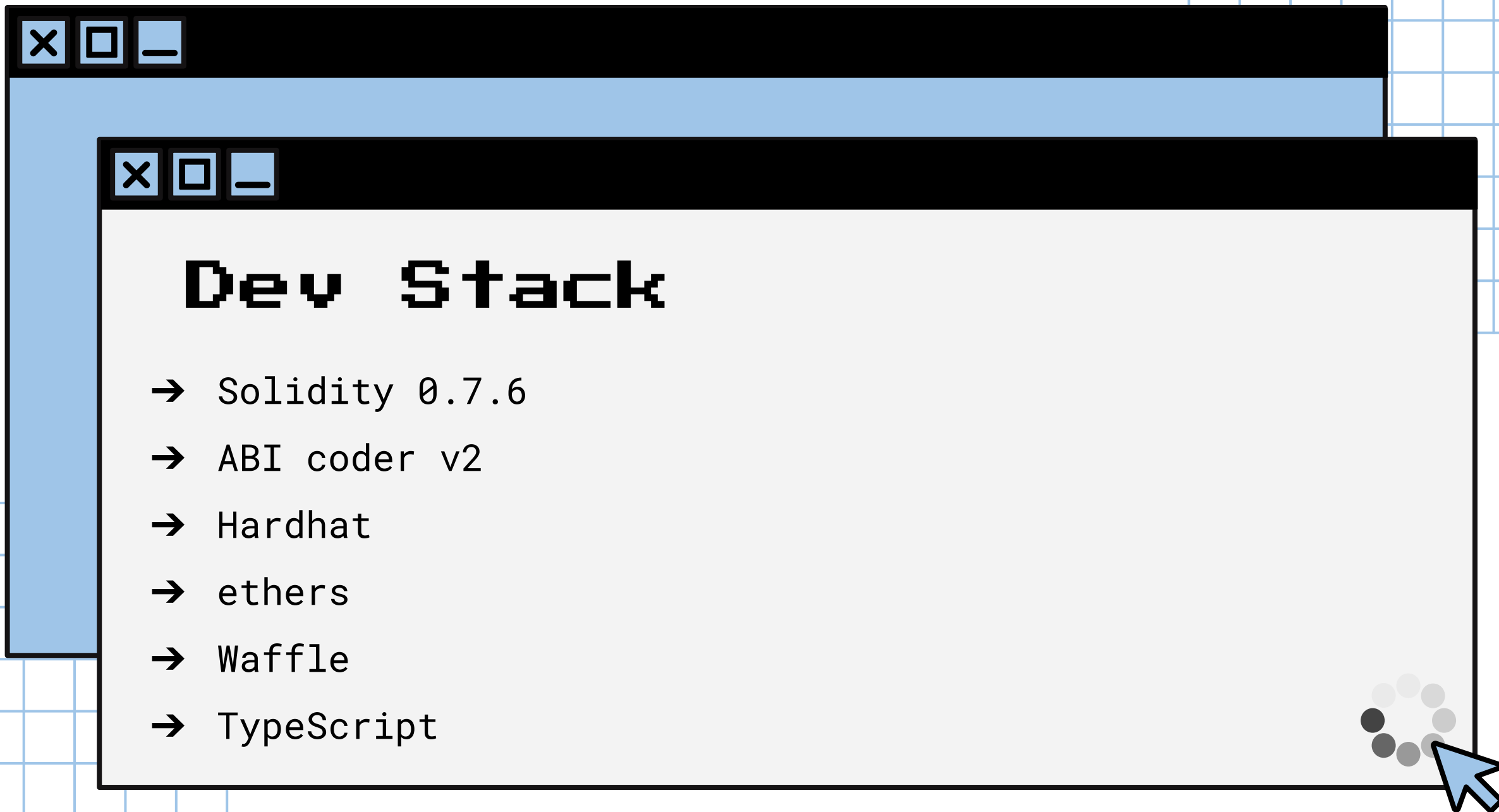
Registry of all
Attestation
Schemas

Simple design based on two main contracts



TL; DR

- Anyone can define an **Attestation Schema (AS)** with a schema and an optional resolver
- Attest to any **AS** 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

Attestation Record

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

```
/**
 * @title A struct representing a record for a submit
 */
struct ASRecord {
    // A unique identifier of the AS.
    bytes32 uuid;
    // Optional schema resolver.
    IASResolver resolver;
    // Auto-incrementing index for reference, assigned
    uint256 index;
    // Custom specification of the AS (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 AS.
 *
 * @param recipient The recipient of the attestation.
 * @param schema The UUID of the AS.
 * @param expirationTime The expiration time of the attestation.
 * @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 schema,
    uint256 expirationTime,
    bytes32 refUUID,
    bytes calldata data
) external payable returns (bytes32);
```

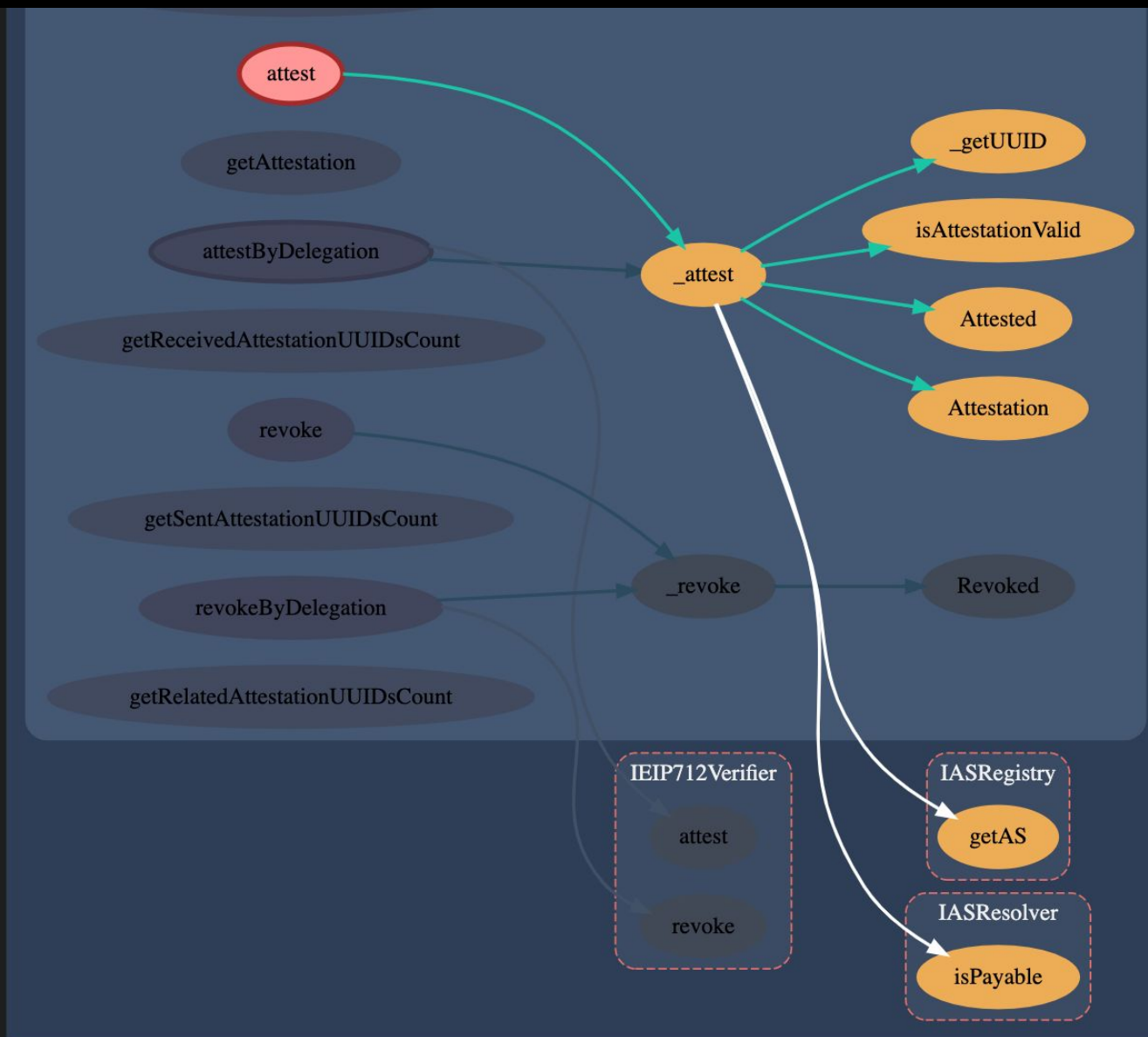
Attest

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**

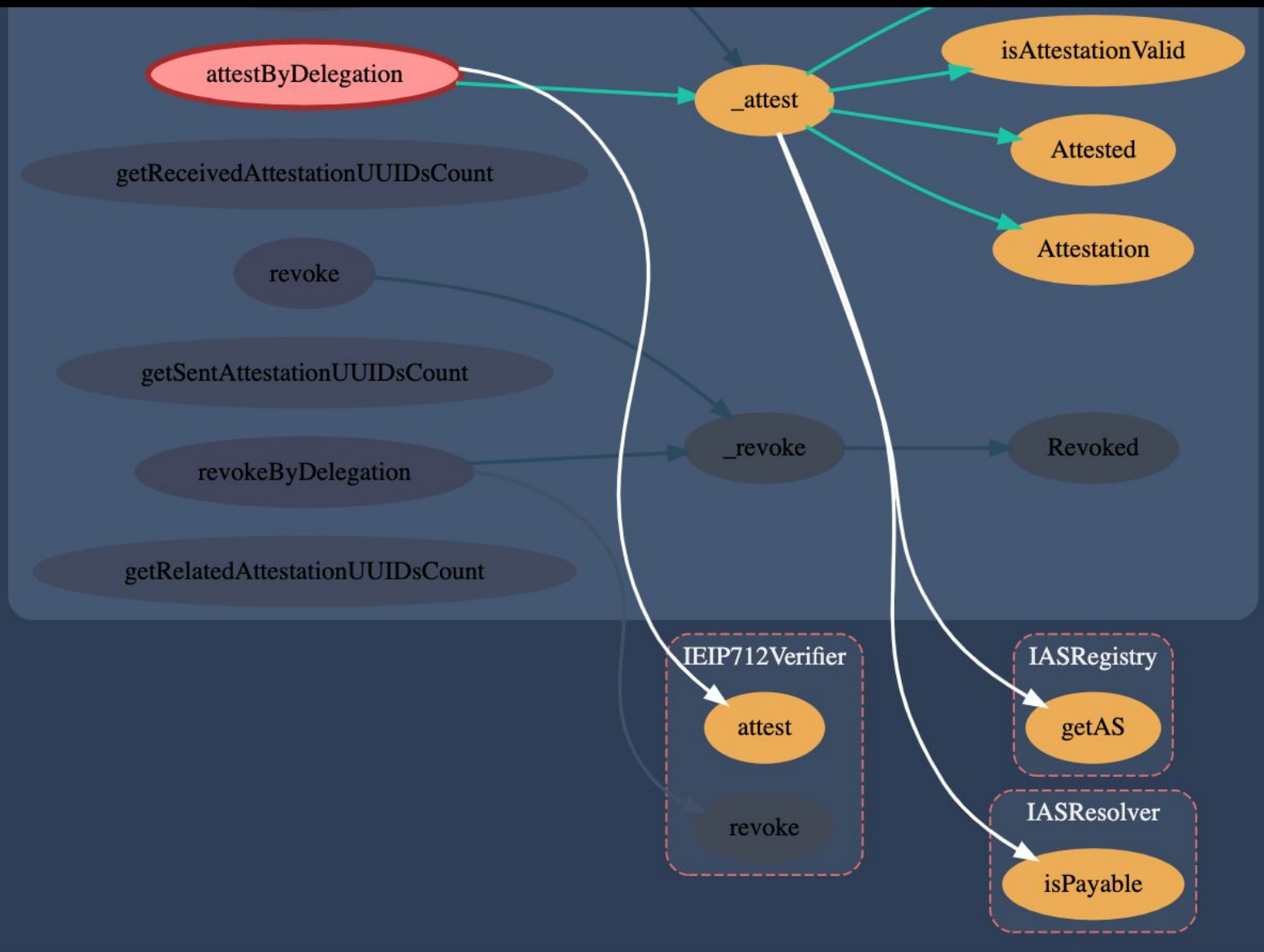
```
/**
 * @dev A struct representing a single attestation.
 */
struct Attestation {
    // A unique identifier of the attestation.
    bytes32 uuid;
    // A unique identifier of the AS.
    bytes32 schema;
    // The recipient of the attestation.
    address recipient;
    // The attester/sender of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



Attest via EIP712

Attesters delegate
their attestation
requests to **msg.sender**
via a EIP712 typed
signature on an
ATTEST_TYPEHASH message

```
/**
 * @dev Attests to a specific AS using a provided EIP712
 *
 * @param recipient The recipient of the attestation.
 * @param schema The UUID of the AS.
 * @param expirationTime The expiration time of the attes
 * @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 schema,
    uint256 expirationTime,
    bytes32 refUUID,
    bytes calldata data,
    address attester,
    uint8 v,
    bytes32 r,
    bytes32 s
) external payable returns (bytes32);
```



```
function attest(
    address recipient↑,
    bytes32 schema↑,
    uint256 expirationTime↑,
    bytes32 refUUID↑,
    bytes calldata data↑,
    address attester↑,
    uint8 v↑,
    bytes32 r↑,
    bytes32 s↑
) external override {
    bytes32 digest = keccak256(
        abi.encodePacked(
            "\x19\x01",
            DOMAIN_SEPARATOR,
            keccak256(
                abi.encode(
                    ATTEST_TYPEHASH,
                    recipient↑,
                    schema↑,
                    expirationTime↑,
                    refUUID↑,
                    keccak256(data↑),
                    nonces[attester↑]++
                )
            )
        )
    );

    address recoveredAddress = ecrecover(digest, v↑, r↑, s↑);
    require(recoveredAddress != address(0) && recoveredAddress == attester↑, "ERR_INVALID_SIGNATURE");
}
```

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,
  schema: string,
  expirationTime: BigNumber,
  refUUID: string,
  data: string,
  nonce: BigNumber,
  privateKey: string
) {
  return this.delegation.getAttestationRequest(
    {
      recipient: typeof recipient === 'string' ? recipient : recipient.address,
      schema,
      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 Verification

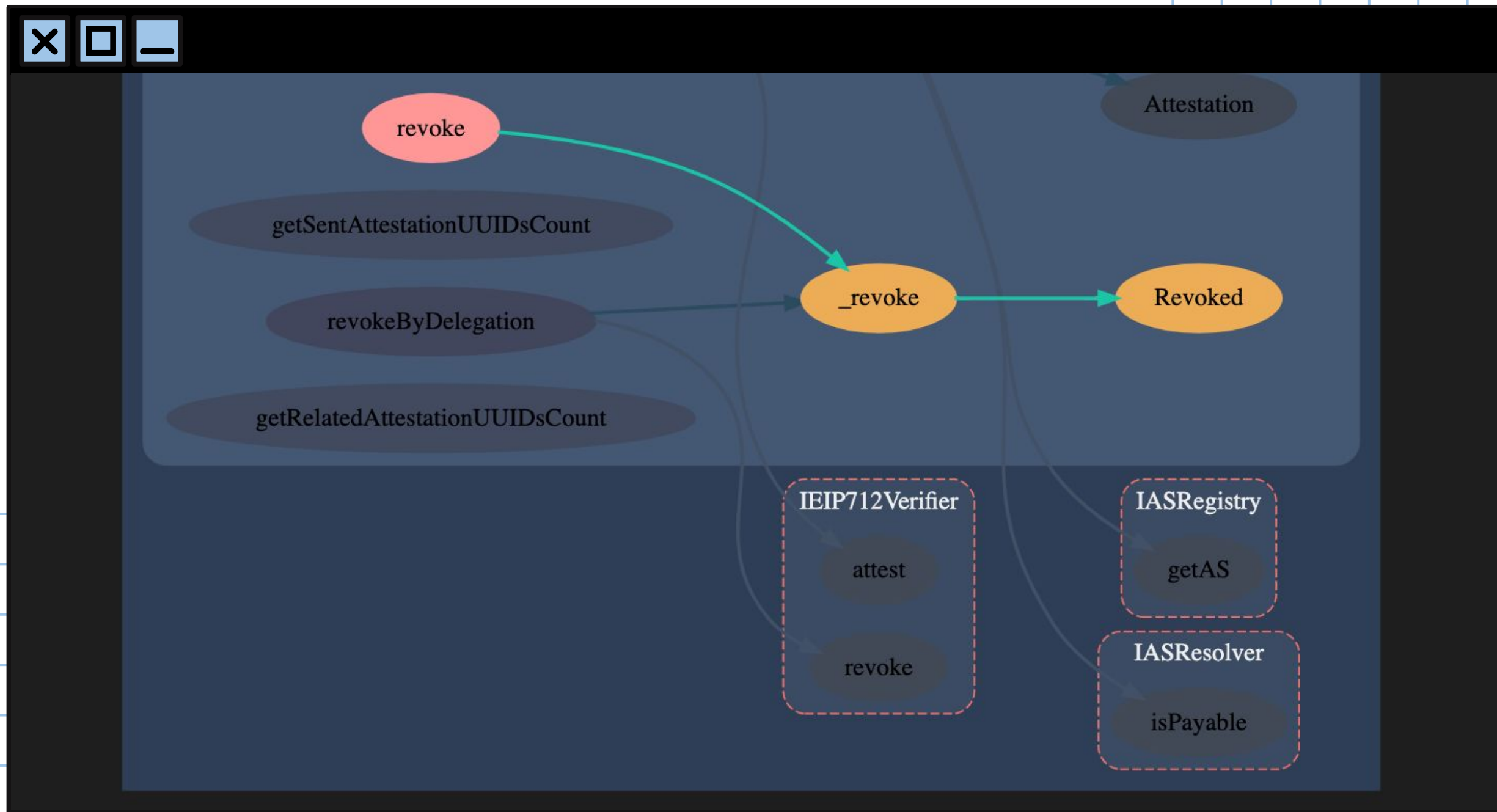
An optional custom schema on-chain verification, set by the registrar of the schema, and is invoked by the **EAS** contract

```
interface IASVerifier {
    /**
     * @dev Verifies whether the specified attestation data conforms to the spec.
     *
     * @param recipient The recipient of the attestation.
     * @param schema The AS 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 message.
     *
     * @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 AS.
 *
 * @param uuid The UUID of the attestation to revoke.
 */
function revoke(bytes32 uuid) external;
```



Revoke via EIP712

Attesters delegate
their revocation
requests to **msg.sender**
via a **EIP712** typed
signature on an
REVOKE_TYPEHASH
message

```
/**
 * @dev Attests to a specific AS using a provided EIP712 si
 *
 * @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;
```




revokeByDelegation

getRelatedAttestationUUIDsCount

_revoke

Revoked

IEIP712Verifier

attest

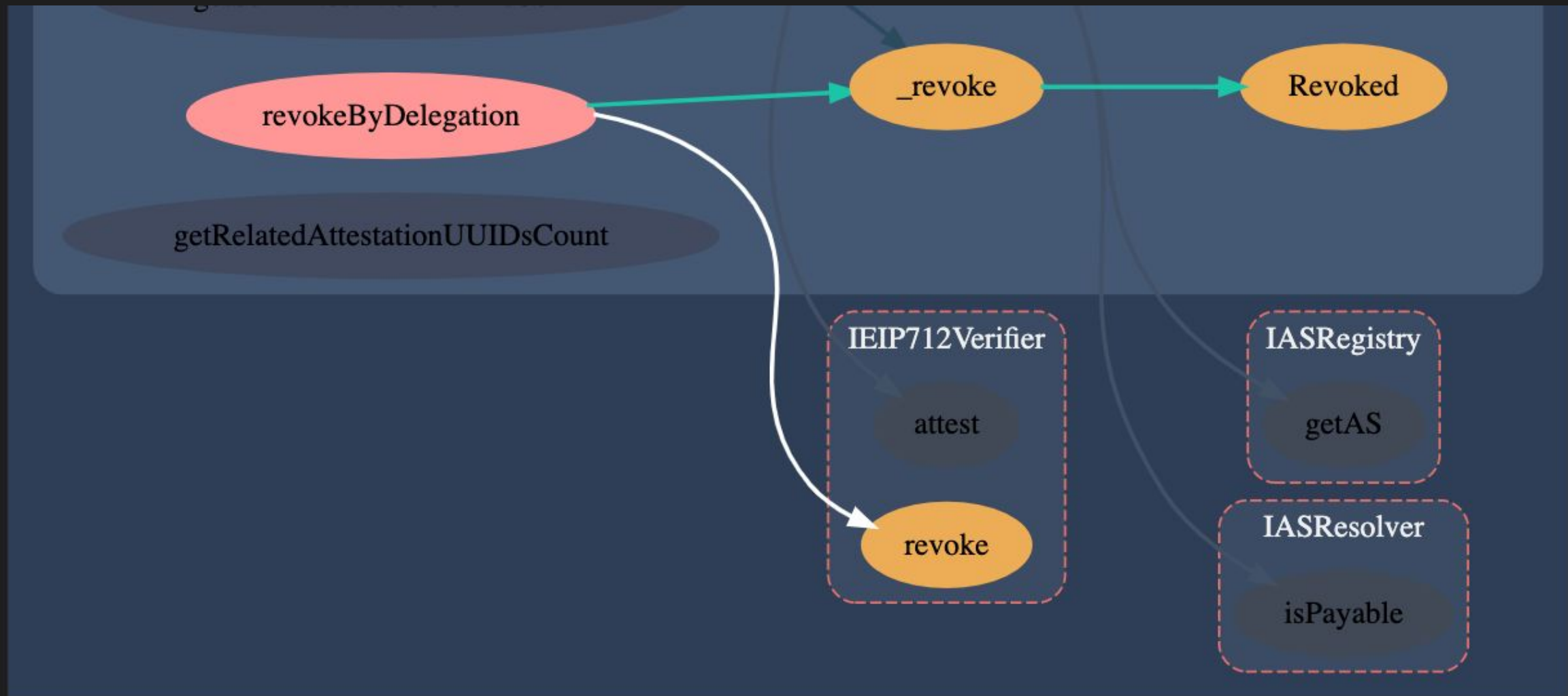
revoke

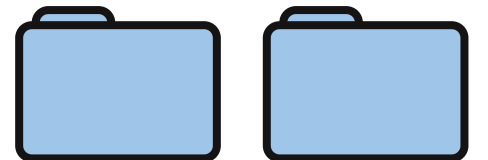
IASRegistry

getAS

IASResolver

isPayable







KYC

Attesting to an identity of an
Ethereum account

Attestation Schema:

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

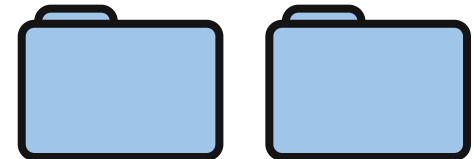
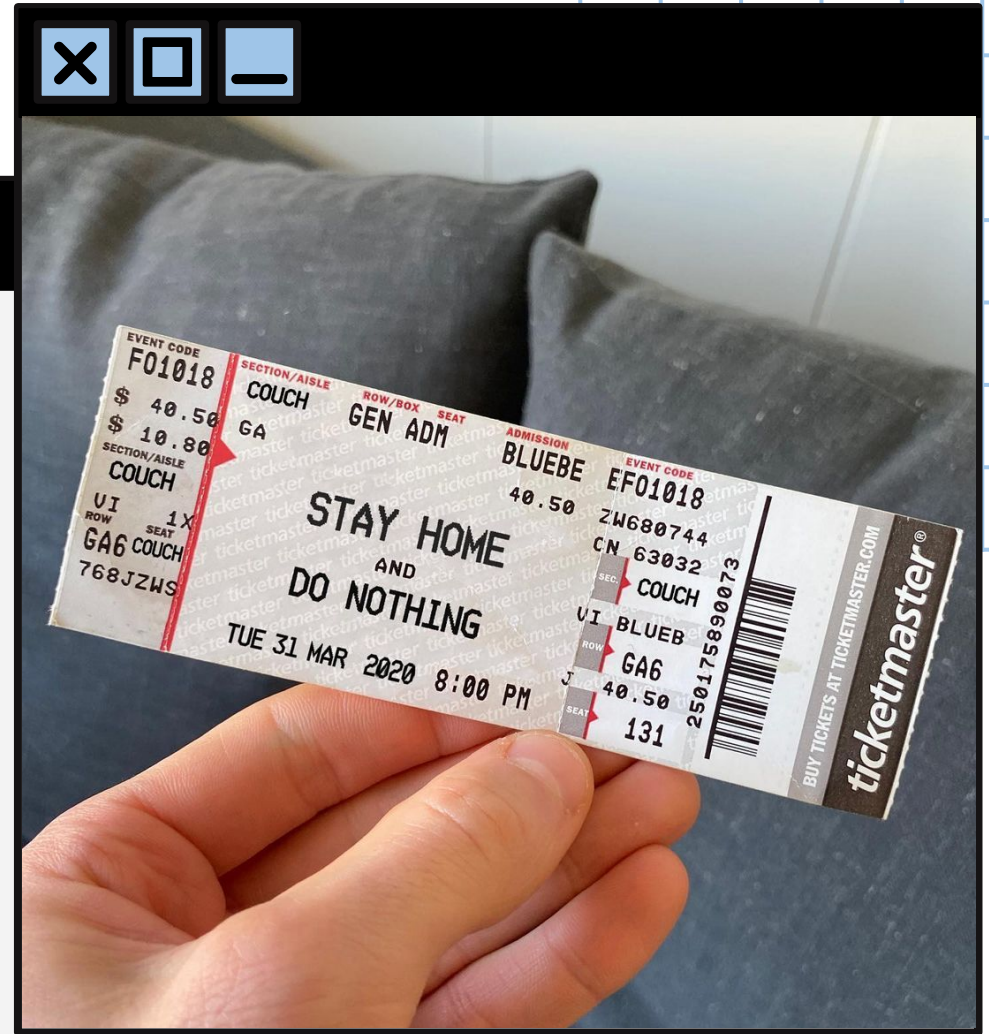


Ticketing System

Attesting to ownership of a ticket

Attestation Schema:

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

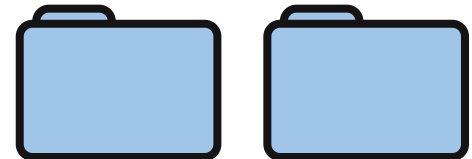
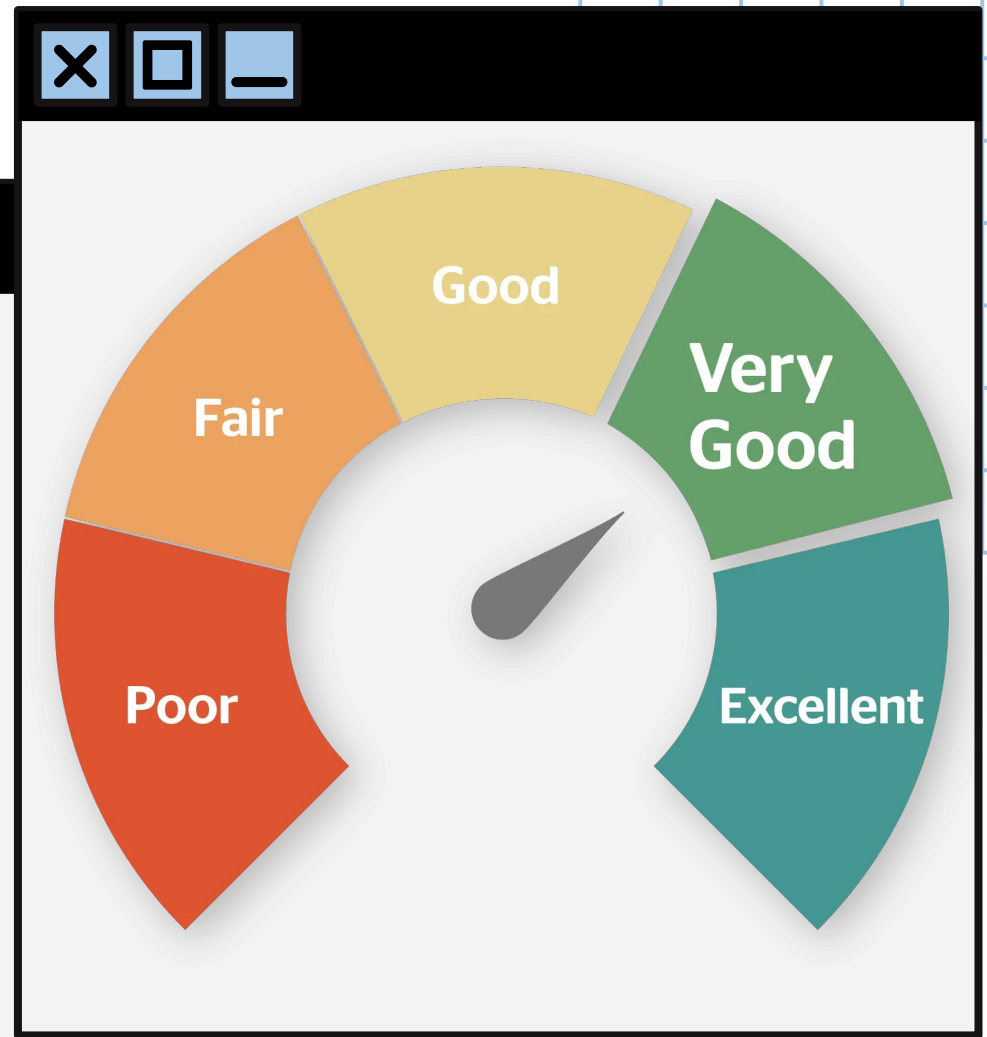


Reputation/ Clout System

Giving a score via an
attestation

Attestation Schema:

(`uint256` rating)



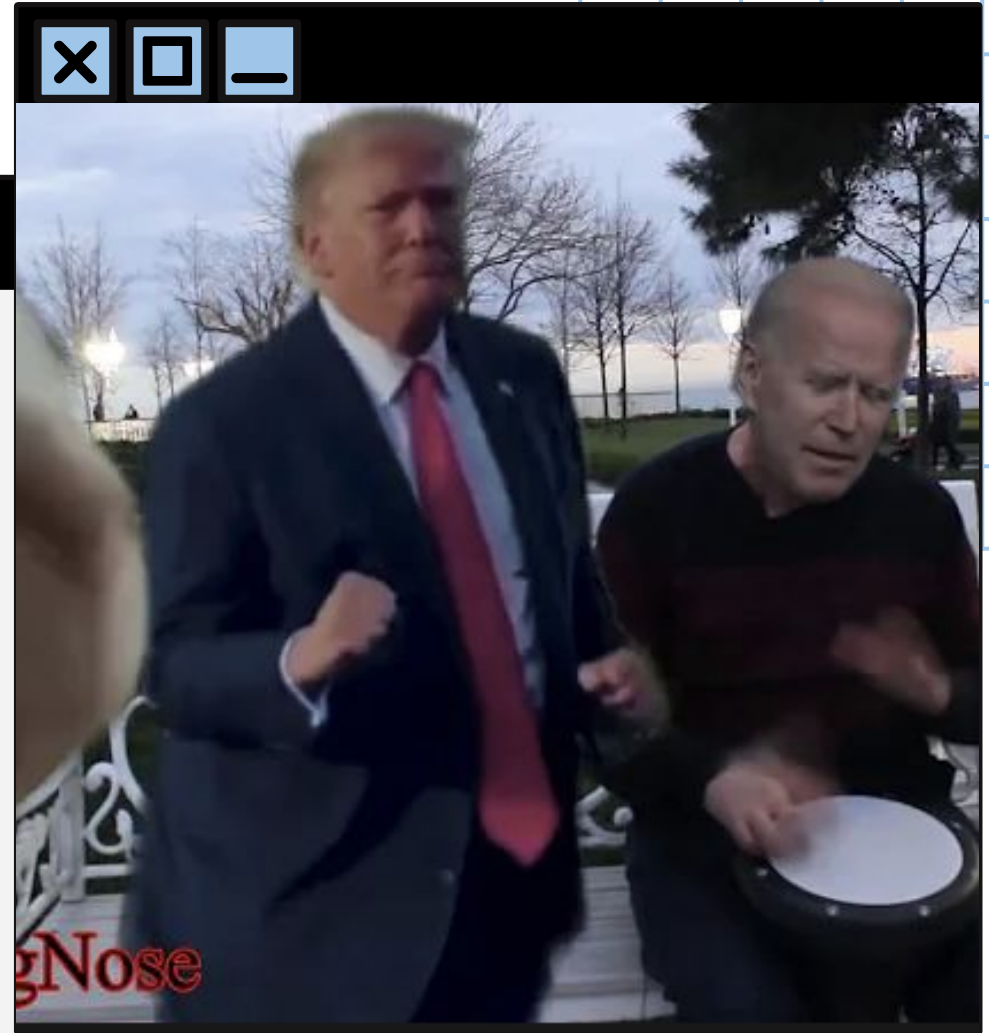


Voting

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

Attestation Schema:

(`bytes32` proposal, `uint8` vote)



Land Registry

A lands authority attests to an ownership of land

Attestation Schema:

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



Oracles

Attest to the outcome real world events.

Attestation Schema:

(**bytes32** realWorldEventID, **bool** outcome)



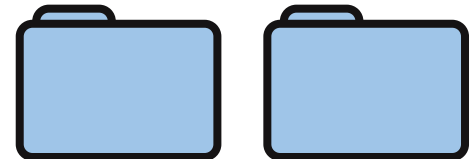
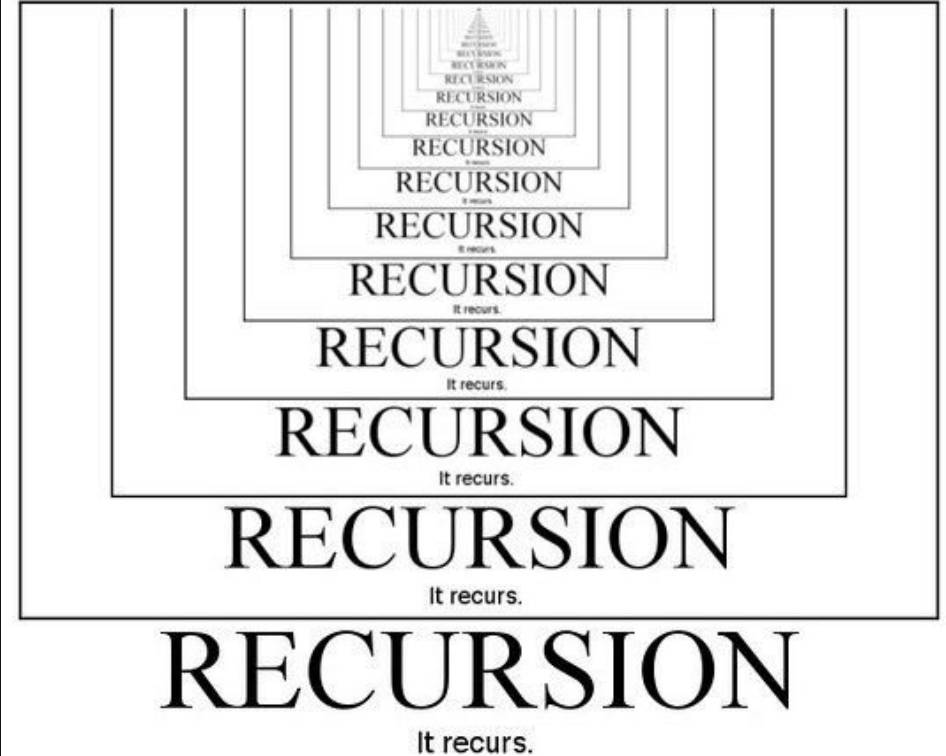


INCEPTION

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

Attestation Schema:

(**bool** trueFalse)



Thank
you!

