1.0 API Stabilizations

Adam Ludvik - Bitwise IO, Minneapolis USA

sawtooth

Transaction Family Creation

Processor SDKs

- Easily define and implement new smart contract languages with Sawtooth Processor SDKs.
- All smart contract languages must be deterministic.

Two things to know:

1. Need to implement `TransactionHandler` interface...
2. ...using a handle to a transaction `Context` object.

Processor SDKs - `TransactionHandler` Interface

- Define Metadata
  - Family Name
  - Family Versions
  - Namespaces

- Implement `TransactionHandler.Apply(Transaction: txn, Context: ctx)`
  - `txn` contains (opaque) payload, signer, and header
  - `ctx` is a handle for specifying the result of executing the transaction
  - Payload interpretation is up to you!

Processor SDKs - Context object

```
getState(List<String> addresses) -> Map<String, Bytes>

setState(Map<String, Bytes> entries) -> List<String>

deleteState(List<String> addresses) -> List<String>


addReceiptData(Bytes data)

addEvent(String type, Map<String, String> attributes, Bytes data)
```

*Note that exact method signatures are language dependent.

Processor SDKs - Example Usage

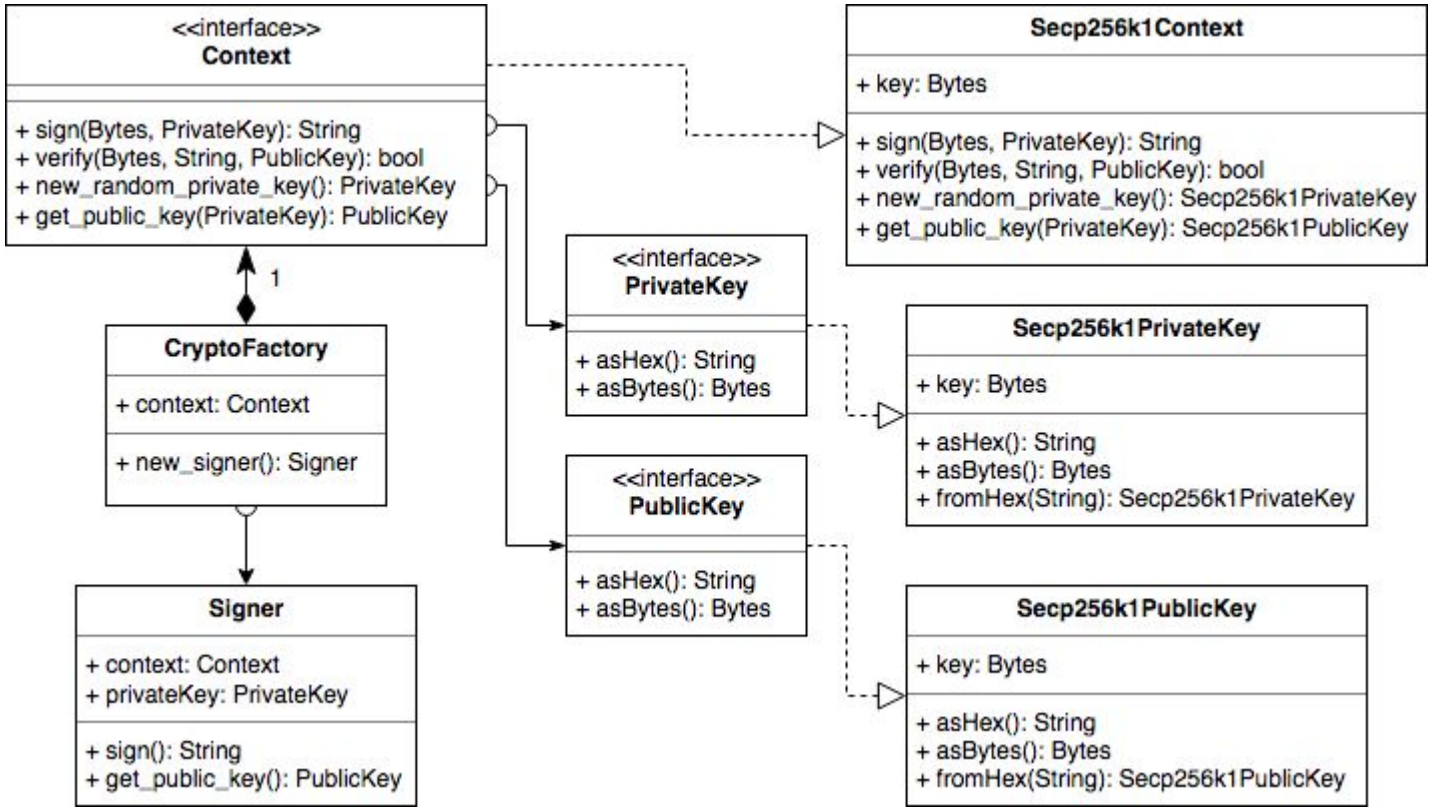
```
function apply(txn: Transaction, ctx: Context) {
    // Decode payload
    payload = parse(txn.payload)
    address = createAddress(payload.unique_id)

    if payload.action == "update_location" {
        // Validate location data and update
        ctx.getState(address)
        ...
        ctx.setState({address: new_location})
    }
    ...
```

Client SDKs

- Simplify implementation of new domain-specific clients.
- Currently, 1.0 API stabilization only includes a signing library.
- Goal of signing library was to define an interface that will support different signing algorithms in the future.

Client SDKs - Signing Library



Client SDKs - Signing Library Usage

```
from sawtooth_signing import create_context
from sawtooth_signing import CryptoFactory

...
context = create_context('secp256k1')
private_key = context.new_random_private_key()
signer = CryptoFactory(context).new_signer(private_key)

...

signer.sign(payload)
```

Interacting with Sawtooth

Client Messages - Two Interfaces

Can interact with a validator node through one of two interfaces

1. REST API - HTTP+JSON
    a. Provides RESTful HTTP routes for writing clients quickly
    b. Less efficient (adapts interface 2)
2. Validator - ZMQ+Protobuf
    a. Provides domain specific request handlers
    b. More efficient

Client Messages - Batches and Transactions

ZMQ+Protobuf	REST
ClientBatchSubmit*	POST /batches
ClientBatchStatus*	GET, POST /batch_statuses
ClientBatchList*	GET /batches
ClientBatchGet*	GET /batches/{batch_id}
ClientTransactionList*	GET /transactions
ClientTransactionGet*	GET /transactions/{transaction_id}

Client Messages - Peers, Blocks, and State

ZMQ+Protobuf	REST
ClientGetPeers*	GET /peers
ClientBlockList*	GET /blocks
ClientBlockGet*	GET /blocks/{block_id}
ClientStateList*	GET /state
ClientStateGet*	GET /state/{address/namespace}
ClientReceiptGet*	GET, POST /receipts

Client Messages - Peers, Blocks, and State

ZMQ+Protobuf	REST
ClientReceiptGet*	GET, POST /receipts
ClientEventsSubsribe*	n/a
ClientEventsUnsubscribe*	
ClientEventsGet*	POST /events

CLI Commands

- `sawtooth` - Main CLI
  - Wraps REST API routes
  - Manage identity
- `sawadm` - Sawtooth Node Administration
  - Generate validator nodes keys
  - Create a new network with a new genesis block
- `sawset` - Manage Sawtooth Settings
  - Create a new settings proposal
  - Vote on proposed settings