Adding Smart Contracts to the XRPL

Timofey Cvetkov

https://github.com/TimCve/rippled-smart-contracts

Smart Contract Goals

Fully Programmable
use real programming language, not just some configurable parameters

Powerful
standalone; can manage assets, keep persistent state & take arguments

Consensus Safe
execution is deterministic, resource bounded and metered at contract deployment to limit spam

Trustless & Secure
immutable code, tightly-restricted application binary interface (ABI) to interact with Rippled

WebAssembly (WASM)

Tiny Footprint
bytes / kbs scale binaries

Fast Execution
near-native performance

Powerful ABI
direct calls to Rippled
C++ method wrappers

Well Understood
compiled from well
recognized languages like C

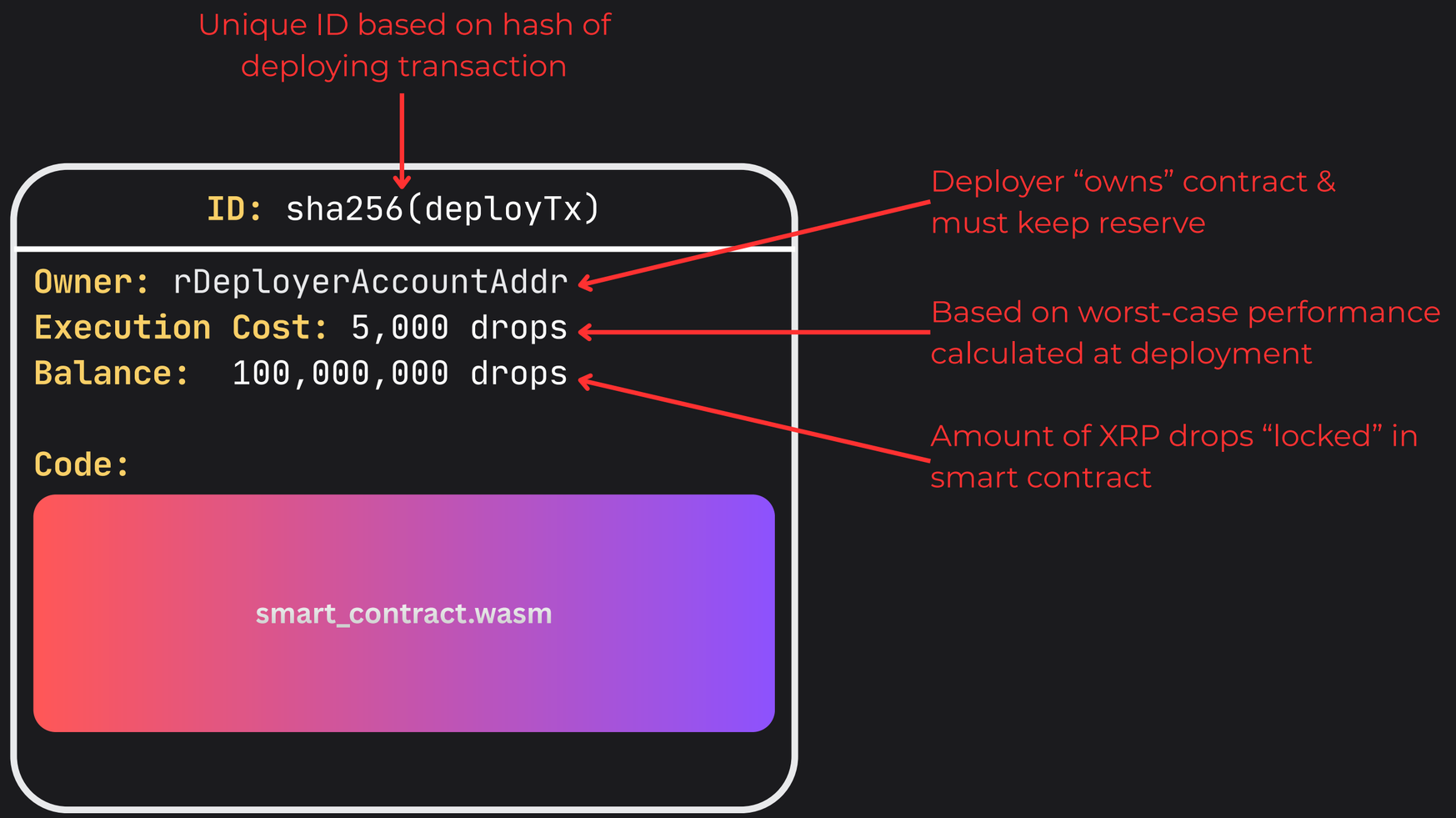
Two New Transactions

ContractDeploy

- validates code
- measures worst-case performance *(determine contract execution fee)*
- inserts contract into ledger

ContractCall

- takes user arguments
- executes logic (WebAssembly)
- reads ledger data
- manages SmartObjects *(general-purpose state)*


Unique ID based on hash of deploying transaction
ID: sha256(deployTx)
Owner: rDeployerAccountAddr
Execution Cost: 5,000 drops
Balance: 100,000,000 drops
Code:
smart_contract.wasm
Deployer “owns” contract & must keep reserve
Based on worst-case performance calculated at deployment
Amount of XRP drops “locked” in smart contract

```c
#include "smart_contract_abi.h"
#include <stdint.h>

typedef struct { uint32_t count; } state_t; // persisted state
typedef struct PACKED { id256_t id; uint32_t delta; } params_t; // call params

__attribute__((export_name("entrypoint")))
int32_t entrypoint(int64_t opt) {
    if (opt == 1) { // init: create counter object
        state_t s = {0}; id256_t id;
        createSmartObject((int32_t)&s, sizeof(s), (int32_t)&id);
        return 0;
    }
    if (opt == 2) { // update: load -> mutate -> store
        if (!paramsPassed()) return -1;
        params_t p;
        if (getParams((int32_t)&p, sizeof(p)) != sizeof(p))
            return -2;
        state_t s;
        if (getSmartObjectData((int32_t)&p.id, (int32_t)&s, sizeof(s)) != sizeof(s))
            return -3;
        s.count += p.delta; setSmartObject((int32_t)&p.id, (int32_t)&s, sizeof(s));
        return 0;
    }
    return -999; // unknown option
}
```

Include Smart Contract ABI

Define data types

Separate “options”

Persistent state management with Smart Objects

SmartObject
-sfSmartObjectID
-sfAccount
-sfContractID
-sfSmartObjectData

- creator (sfAccount) must keep reserve
- SmartObject operations have definite costs (100 ~ 150 drops)

every ABI function has a definite cost (all included in transaction fee)

```c
#include "smart_contract_abi.h"
#include <stdint.h>

typedef struct { uint32_t count; } state_t; // persisted state
typedef struct PACKED { id256_t id; uint32_t delta; } params_t; // call params

__attribute__((export_name("entrypoint")))
int32_t entrypoint(int64_t opt) {
    if (opt == 1) { // init: create counter object
        state_t s = {0}; id256_t id;                    -0.00015 XRP
        createSmartObject((int32_t)&s, sizeof(s), (int32_t)&id);
        return 0;
    }
    if (opt == 2) { // update: load -> mutate -> store
        if (!paramsPassed()) return -1;
        params_t p;          -0.00001 XRP
        if (getParams((int32_t)&p, sizeof(p)) != sizeof(p))
            return -2;                -0.00003 XRP
        state_t s;                                                  -0.0001 XRP
        if (getSmartObjectData((int32_t)&p.id, (int32_t)&s, sizeof(s)) != sizeof(s))
            return -3;
        s.count += p.delta; setSmartObject((int32_t)&p.id, (int32_t)&s, sizeof(s));
        return 0;                                                   -0.00015 XRP
    }
    return -999; // unknown option
}
```

XRP Managment

Only contract owner and contract caller XRP can be “locked”

XRP unlocked from contract balance

```c
#include "smart_contract_abi.h"
#include <stdint.h>

#define AMOUNT 100

/* entrypoint definition, logic, etc. */

// XRP "locking"
lockOwnerXRP(AMOUNT);  // ContractBalance += 100
lockCallerXRP(AMOUNT); // ContractBalance += 100
```

contract balance = 200 drops

```c
addr_t owner;
getOwnerAddr((int32_t)&owner);

int64_t bal;
bal = getContractBalance();

// XRP "unlocking"
unlockXRP(bal, (int32_t)&owner); // ContractBalance -= 200
                                 // Owner Balance += 200
```

contract balance = 0 drops

escrow-style scenarios
i.e. locking funds for later unlocking must be manually managed via SmartObjects

Loop Guards

inspired by XRPL Hooks proposal

The Smart Contract ABI provides a mandatory loop guard function.

```c
#include "smart_contract_abi.h"
#include <stdint.h>

// automatically set guard ID to line number
#define GUARD(maxiter) _g(__LINE__, maxiter)

/* entrypoint defintion, logic, etc. */

for (uint32_t i = 0; i < 100; i++) {
    GUARD(100); // _g(10, 100)

    for (uint32_t j = 0; j < 100; j++) {
        GUARD(10000); // _g(13, 10000)
    }
}
```

Two Purposes:

- guideline for worst-case loop performance during code metering at deployment
- spam prevention by hard stopping execution if guard exceeded

A real application...

Trustless coin flip game via smart contracts on the XRPL

The Game Premise

The Game Premise

DEALER

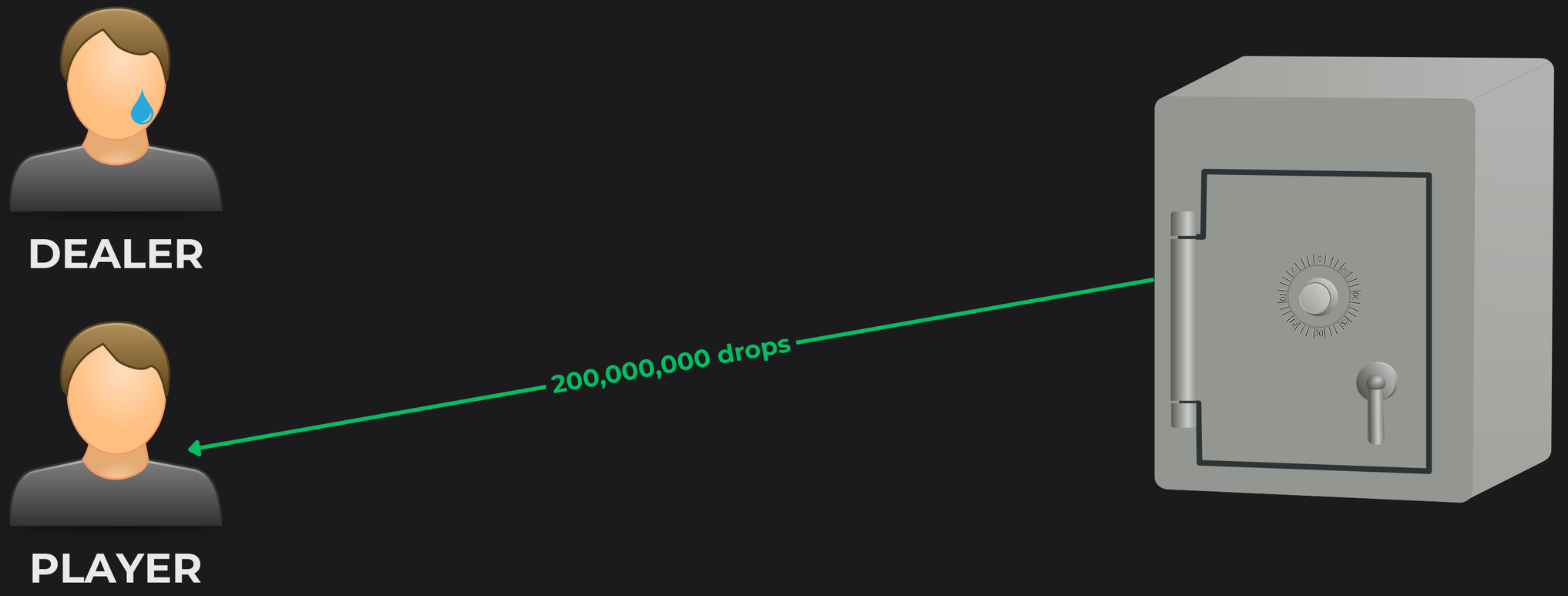
```
secret8 = 00000001b
salt256 = 604..8ea
entry = sha256(secret||salt)
```

PLAYER

```
entry8 = 00000001b
betDrops = 100,000,000
```

```
00000001b == 00000001b
```

The Game Premise




```
00000001b == 00000001b
```

From the technical side...

Deployed SmartContract Object:

```
{
  "Account": "rHb9CJAWyB4rj91VRWn96DkukG4bwdtyTh", # owner address
  "ContractCode": "006...20B", # coinflip.wasm (2793 bytes)
  "ContractCost": "828", # derived from worst-case performance at deployment
  "ContractID": "BEF51CFD39116BED736B91C943FE03338152B3CEF00E917D789CA29C8F17794D"
}
```

ContractCall Tx - Dealer Proposes Game:

```
# secret: 01
# salt: 604af710926e6c9a3fbe5cb21f074a3d0f3107e055adcefca08bc9909f3a28ea

{
  "TransactionType": "ContractCall",
  "Account": "rHb9CJAWyB4rj91VRWn96DkukG4bwdtyTh", # genesis account
  "ContractID": "BEF51CFD39116BED736B91C943FE03338152B3CEF00E917D789CA29C8F17794D",
  "ContractOpt": "1",
  "ContractParams": "d1b848f4316e48099a98470aa759aceee3a1921e79148c051bb576660c71d02d"
} # signed with masterpassphrase                                    sha256(secret||salt)
```


ContractOpt is an efficient, numerical parameter
that any ContractCall can take

Game State SmartObject (created by previous contract call):

```
# owned by dealer (OwnerCount++)

{
  "Account": "rHb9CJAWyB4rj91VRWn96DkukG4bwdtyTh",
  "ContractID": "BEF51CFD39116BED736B91C943FE03338152B3CEF00E917D789CA29C8F17794D",
  "SmartObjectData":                                    dealer commit
"d1b848f4316e48099a98470aa759aceee3a1921e79148c051bb576660c71d02d0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000",
  "SmartObjectID": "96253C9B267278412F633CACA9B5733BACD6FF68E76674E5EFDBFC5787051BEB"
}
```

SmartObjectData: { commit256, accepted8, timestamp32,
player160, player_entry8, player_bet64 }

ContractCall Tx - Player Accepts Game:

```
{
  "TransactionType": "ContractCall",
  "Account": "rBdZfji6NiSaZK4QNphKzjvQddymPf3tib", # player account
  "ContractID": "BEF51CFD39116BED736B91C943FE03338152B3CEF00E917D789CA29C8F17794D",
  "ContractOpt": "2",
  "ContractParams":
"96253C9B267278412F633CACA9B5733BACD6FF68E76674E5EFDBFC5787051BEB0100e1f50500000000"
} # signed with ssSfFoCVfb3AGMuVJccmwWWMD35E3
```

Game ID (previous SmartObjectID)

player bet (100 XRP)

player entry (same as dealer)

Game State SmartObject (updated by previous contract call):

```
# owned by dealer (still)

{
  "Account": "rHb9CJAWyB4rj91VRWn96DkukG4bwdtyTh",
  "ContractID": "BEF51CFD39116BED736B91C943FE03338152B3CEF00E917D789CA29C8F17794D",
  "SmartObjectData":
"d1b848f4316e48099a98470aa759aceee3a1921e79148c051bb576660c71d02d01000000 4AA72831 749B0AA
D3D2D0CF32E128BD8DB3B9D4B00C74E0F 01000000 00E1F50500000000",
  "SmartObjectID": "96253C9B267278412F633CACA9B5733BACD6FF68E76674E5EFDBFC5787051BEB"
}
```

commit256 accepted8 timestamp32

player160 player_entry8 player_bet64

SmartObjectData: { commit256, accepted8, timestamp32,
player160, player_entry8, player_bet64 }

ContractCall Tx - Dealer Reveals Secret & Salt:

```
# secret: 01
# salt: 604af710926e6c9a3fbe5cb21f074a3d0f3107e055adcefca08bc9909f3a28ea

{
  "TransactionType": "ContractCall",
  "Account": "rHb9CJAWyB4rj91VRWn96DkukG4bwdtyTh",
  "ContractID": "BEF51CFD39116BED736B91C943FE03338152B3CEF00E917D789CA29C8F17794D",
  "ContractOpt": "3",
  "ContractParams":                                  Game ID
"96253C9B267278412F633CACA9B5733BACD6FF68E76674E5EFDBFC5787051BEB01604af710926e6c9a3fbe5
cb21f074a3d0f3107e055adcefca08bc9909f3a28ea"                    revealed secret
} # signed with masterpassphrase    revealed salt
```

Outcome: Since dealer and player submitted same entry, player wins 200 XRP 🎉

Game State SmartObject:

```
# owned by dealer

{
  "Acco   ": "r   cJAWyB4rj91VRWn96Dku   hwdtyT   ",
  "Contrac   "BEF51CFD39116BED736B91C9   338152B3CEF00E917D789CA29C   794D",
  "SmartOb   ta":
"d1b848   16e480   8470aa759aceee3a19   79148c   b576660c71d02d010   504AA728   49B0AA
D3D2   F32E128BD8DB   4B00C74E0F01   00000E1F5050   000",
  "SmartObjectID": "962   B26727   12F633CACA9B5733BACD   68E764   E5EFDBFC5787051BEB"
}
```

Game State SmartObject is REMOVED

What if a crooked dealer doesn’t reveal their secret & salt?

The timestamp saved when player accepts the game allows the player to claim all staked funds after 24 hours (win case)!

(also part of the smart contract, not a built-in feature)

Goals Achieved?

Fully Programmable
use real programming language, not just some configurable parameters

Powerful
standalone; can manage assets, keep persistent state & take arguments

Consensus Safe
execution is deterministic, resource bounded and metered at contract deployment to limit spam

Trustless & Secure
immutable code, tightly-restricted application binary interface (ABI) to interact with Rippled

Potential Future Work

- Move smart contracts to pseudo-accounts (XLS-0064)
- Allow owner to delete contract if contract balance is empty
- Separate fees for different options/functions of a contract (derived from worst-case performance of each one)
- Allow inter-contract interaction e.g. via shared SmartObjects or cross-contract function imports

Hope you enjoyed my presentation!

Timofey Cvetkov

https://github.com/TimCve/rippled-smart-contracts