XRPL Smart Contracts

Author: Timofey Cvetkov

Smart Contracts on the XRPL are written in WebAssembly (WASM) and use a custom Smart Contract ABI (Application Binary Interface) to interact with the ledger via imported functions. All ABI imports (functions, data types, macros) are imported into the smart contract code through the smart_contract_abi.h header file.

New Dependencies

Binaryen

- Used for parsing compiled WebAssembly code during smart contract deployment
- Git submodule in /external/binaryen

Wasmtime

- Used for executing WebAssembly code - powers the smart contract WASM VM
- Conan package

New Transaction Types

ContractDeploy

Name	Type
sfContractCode	VL

ContractDeploy::preflight

- Reject if smart contracts amendment is off: temDISABLED
- Reject any universal flags: temINVALID_FLAG
- Run shared preflight1 to check core tx shape (account, fee format, key format, ticket/accountTxnID conflicts, etc.)
- Require non empty sfContractCode, else temMALFORMED
- Run preliminary smart contract validation:
    - Only allow function imports defined in smart_contract_abi.h
    - Ensure “entrypoint” exists and is the only exported function
    - Ensure no banned WebAssembly instructions/tokens are present (e.g. memory.grow, call_indirect, f32 & f64)
    - Ensure no recursion by building a direct internal call graph and rejecting recursion cycles

  Returns temMALFORMED if any preliminary check fails
- Run shared preflight2 to check signature validity

ContractDeploy::calculateBaseFee

Fee = normal transactor base fee + 1 drop (0.000001 XRP) per byte of sfContractCode

ContractDeploy::doApply

By this point, shared Transactor::apply already captured mPriorBalance, consumed seq/ticket, and charged fee.

- Load owner (deployer) account SLE; if missing then tefINTERNAL
- Reserve check for one additional owned object; if mPriorBalance < reserve, return tefINSUFFICIENT_RESERVE
- Derive contractID as sha512Half(txID) and build keylet::smartContract(contractID)
- Create smart contract SLE and set sfAccount and sfContractID fields
- Run smart contract code metering (worst-case performance):
    - Load contract code (WebAssembly) with Binaryen (ModuleHandle) and validate module before analysis
    - Resolves canonical exported “entrypoint” function to analyze
    - Builds Analyzer with cost model (base=1, internalCallOverhead=2, individually specified, fixed costs for each smart_contract_abi.h import call)
    - The analyzer computes costFunctionWorst(entrypoint) using memoization and recursion-marking (White/Gray/Black) for call graph traversal
        - Memoization caches previously computed results so they are not recomputed every time, i.e. calculated worst case performance for a given function is cached
        - Recursion-marking tracks DFS visit state to detect cycles - White (unvisited), Gray (currently in call stack), Black (done). If analysis reaches a Gray function again, that indicates recursion, so it throws an error
        - Note: preflight already checks for recursion via a call-graph scan, call graph traversal in doApply shouldn’t ever find recursion
    - For each WASM expression, it accumulates node + operand/child costs; function calls are charged as:
      imported call = fixed host cost,
      internal call = internalCallOverhead + worst(callee)
    - Control-flow is metered conservatively by taking worst-case branches for if/select/switch
    - Loops must be explicitly guarded: first non-nop must call host._g(..., i32.const max_iters); analyzer extracts max_iters and scales loop continuation cost by that bound

  Computed cost stored in sfContractCost field of smart contract SLE
- Contract code stored in sfContractCode field of smart contract SLE
- Smart contract balance (sfContractBalance) set to 0 initially
- Smart contract is inserted into the ledger as a ledger object and then inserted into the deployer’s owner directory

ContractCall

Name	Type
sfContractID	UINT256
sfContractParams	VL
sfContractOpt	UINT64

The idea is that sfContractOpt serves as an efficient numerical parameter to differentiate between “options” inside the smart contract code. The sfContractParams field is an arbitrary length, arbitrary format field which must be parsed manually inside the smart contract code (by means of a struct or simple data type).

ContractCall::preflight

- Reject if smart contracts amendment is off: temDISABLED
- Reject any universal flags: temINVALID_FLAG
- Run shared preflight1 to check core tx shape (account, fee format, key format, ticket/accountTxnID conflicts, etc.)
- Run shared preflight2 to check signature validity

ContractCall::preclaim

- Shared preclaim checks run first (seq/ticket, prior-tx/last-ledger, fee, permission, sign)
- ContractCall::preclaim verifies the contract SLE exists; else tecNO_ENTRY
- Verifies caller account exists; else terNO_ACCOUNT

ContractCall::calculateBaseFee

Fee = normal transactor base fee + sfContractCost (calculated during ContractDeploy)

ContractCall::doApply

- Loads contract by sfContractID; missing gives tecNO_ENTRY
- Reads wasm code from sfContractCode; empty gives tecFAILED_PROCESSING
- Ensures sfContractBalance exists (initializes to 0 if absent)
- Builds runtime HostState (caller, owner, params, opt, ledger view)
- Compiles module with Wasmtime; compile/engine errors map to tecFAILED_PROCESSING
- Enforces runtime import/export policy (only approved host.* imports; function export must be “entrypoint”
- Registers all host callbacks (getCallerAddr, lockCallerXRP, createSmartObject, _g, etc.)
- Instantiates module, verifies memory export, locates entrypoint, then calls it with one i64 argument (sfContractOpt) and expects one i32 result
- If Wasm traps and a host callback already set st.callbackTer, that callback TER is returned (preserved)
- If no callback TER but Wasm fails, returns tecFAILED_PROCESSING
- If call succeeds but result i32 != 0, returns tecFAILED_PROCESSING

SmartObject

SmartObject is a new ledger object type designed specifically for use with smart contracts:

Name	Type
sfSmartObjectID	UINT256
sfAccount	ACCOUNT
sfContractID	UINT256
sfSmartObjectData	VL
sfOwnerNode	UINT64
sfPreviousTxnID	UINT256
sfPreviousTxnLgrSeq	UINT32

SmartObject interaction is performed only through smart contract code, specifically smart_contract_abi.h imports:

- createSmartObject -> void
- getSmartObject -> {owner, data}
- getSmartObjectData -> {data}
- deleteSmartObject -> void

SmartObject data can be of arbitrary size and format. Similarly to sfContractParams, it must be parsed manually inside the smart contract code (by means of a struct or simple data type).

RNG/Gambling Application

Traditional RNG (Random Number Generation) is useless on a replicated ledger system like the XRPL because it's not possible to hide the seed. By knowing the seed, any user could easily predict any given pseudo-random number, limiting applications like gambling.

However, a different approach to RNG, managed via a smart contract could be the solution to gambling (and other RNG-reliant applications) on the XRPL. Such an approach relies on multiple users committing secrets using cryptographic hashes and revealing them during a specified reveal window under threat of an economic penalty in case of failing to reveal.

Lottery Smart Contract Example

Overview

- Prize pool: 250 XRP
- Entry fee: 5 XRP
- 100 participants

Commit Round:

- commit window opens for 24 hours
- each participant makes a commit: sha256(AccountID, ContractID, SECRET_64) (commits stored in "paginated linked list" implemented via ledger objects)
- to submit a commit, each participant locks 5 XRP entry fee + 50 XRP deposit
- commit window closes

Reveal Round:

- reveal window opens for 24 hours
- each participant reveals their secret
- revealed secret is verified against commit
- if revealed secret matches, participant is refunded 50 XRP deposit
- if revealed secret is incorrect, nothing happens (deposit stays in contract balance)
- reveal window closes

Pseudo-Random RNG:

- all revealed secrets are XORed together to get winning value
- all revealed secrets are XORed against winning value and smallest resulting value wins prize pool (if 2 identical secrets were committed by two participants, earlier one wins)

Smart Contract Implementation

Compiled bytecode size: 3955 bytes
Computed contract cost: 0.164316 XRP

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

#define GUARD(maxiter) _g(__LINE__, maxiter)

#define TRUE 1
#define FALSE 0

#define PRIZE 250000000        // 250 XRP
#define ENTRY_DEPOSIT 50000000 // 50 XRP
#define ENTRY_FEE 5000000      // 5 XRP
#define PARTICIPANT_PAGE_COUNT 10
#define PARTICIPANT_PAGE_CAPACITY 10 // 10 x 10 = 100 total
participants
#define COMMIT_WINDOW_DURATION 24 * 60 * 60 // 24 hours
#define REVEAL_WINDOW_DURATION 24 * 60 * 60 // 24 hours
```

```
// Runtime participant state is intentionally NOT packed so hash256_t
remains
// naturally aligned for direct 64-bit word access.
typedef struct {
   addr_t address;
   uint8_t revealed;
   hash256_t entry; // commit (all 256 bits) / revealed secret (first
64 bits)
} participant_t;

// Runtime page state is intentionally NOT packed for the same
alignment reason.
typedef struct {
   uint32_t filled_slots;
   participant_t participants[10];
   id256_t next_page_id;
} participant_page_t;

// Runtime lottery state is intentionally NOT packed.
typedef struct {
   uint32_t start_timestamp;
   id256_t first_participant_page_id;
   uint64_t winning_value;
   uint8_t winning_value_determined;
} lottery_t;

// Packed on purpose: this exact byte layout is hashed for commit
verification.
typedef struct PACKED {
   addr_t account_id;
   id256_t contract_id;
   uint64_t secret;
} commit_unhashed_t;

// Runtime winner state is intentionally NOT packed.
typedef struct {
   addr_t account_id;
   uint64_t difference;
} winner_t;

// Packed on purpose: params are serialized bytes coming from
transaction input.
typedef struct PACKED {
```

```
   id256_t lottery_id;
   hash256_t commit;
} params2_t;

// Packed on purpose: params are serialized bytes coming from
transaction input.
typedef struct PACKED {
   id256_t lottery_id;
   uint64_t secret;
} params3_t;

// Size and alignment checks guard against layout drift and unaligned
uint64_t access.
_Static_assert(sizeof(participant_t) == 56, "participant_t size
mismatch");
_Static_assert(sizeof(participant_page_t) == 600, "participant_page_t
size mismatch");
_Static_assert(sizeof(lottery_t) == 56, "lottery_t size mismatch");
_Static_assert(sizeof(commit_unhashed_t) == 60, "commit_unhashed_t size
mismatch");
_Static_assert(sizeof(winner_t) == 32, "winner_t size mismatch");
_Static_assert(sizeof(params2_t) == 64, "params2_t size mismatch");
_Static_assert(sizeof(params3_t) == 40, "params3_t size mismatch");
_Static_assert(
   _Alignof(hash256_t) >= _Alignof(uint64_t),
   "hash256_t alignment must allow uint64_t access");
_Static_assert(
   (offsetof(participant_t, entry) % _Alignof(uint64_t)) == 0,
   "participant_t.entry must be uint64_t-aligned");

int32_t hash256_eq(hash256_t* a, hash256_t* b) {
   if (a->words[0] == b->words[0] &&
       a->words[1] == b->words[1] &&
       a->words[2] == b->words[2] &&
       a->words[3] == b->words[3])
       return TRUE;
   else return FALSE;
}

int32_t addr_eq(addr_t* a, addr_t* b) {
   if (((uint8_t*)a)[0] == ((uint8_t*)b)[0] && ((uint8_t*)a)[1] ==
((uint8_t*)b)[1] &&
```

```
       ((uint8_t*)a)[2] == ((uint8_t*)b)[2] && ((uint8_t*)a)[3] ==
((uint8_t*)b)[3] &&
       ((uint8_t*)a)[4] == ((uint8_t*)b)[4] && ((uint8_t*)a)[5] ==
((uint8_t*)b)[5] &&
       ((uint8_t*)a)[6] == ((uint8_t*)b)[6] && ((uint8_t*)a)[7] ==
((uint8_t*)b)[7] &&
       ((uint8_t*)a)[8] == ((uint8_t*)b)[8] && ((uint8_t*)a)[9] ==
((uint8_t*)b)[9] &&
       ((uint8_t*)a)[10] == ((uint8_t*)b)[10] && ((uint8_t*)a)[11] ==
((uint8_t*)b)[11] &&
       ((uint8_t*)a)[12] == ((uint8_t*)b)[12] && ((uint8_t*)a)[13] ==
((uint8_t*)b)[13] &&
       ((uint8_t*)a)[14] == ((uint8_t*)b)[14] && ((uint8_t*)a)[15] ==
((uint8_t*)b)[15] &&
       ((uint8_t*)a)[16] == ((uint8_t*)b)[16] && ((uint8_t*)a)[17] ==
((uint8_t*)b)[17] &&
       ((uint8_t*)a)[18] == ((uint8_t*)b)[18] && ((uint8_t*)a)[19] ==
((uint8_t*)b)[19])
       return TRUE;
   else return FALSE;
}

__attribute__((export_name("entrypoint")))
int32_t entrypoint(int64_t opt) {

   addr_t contract_owner;
   addr_t contract_caller;
   id256_t contract_id;
   getOwnerAddr((int32_t)&contract_owner);
   getCallerAddr((int32_t)&contract_caller);
   getContractId((int32_t)&contract_id);

   switch(opt) {
       case 1: // initialize lottery
       {
           if (!addr_eq(&contract_caller, &contract_owner)) return -11;

           // 1 lottery at a time restriction
           // when lottery is finalized, contract balance goes to 0
           if (getContractBalance() > 0) return -12;

           lockOwnerXRP(PRIZE);
```

```
            lottery_t lottery;
            lottery.start_timestamp = getLedgerTimestamp();

            participant_page_t page;
            page.filled_slots = 0;
            for (uint32_t i = 0; i < PARTICIPANT_PAGE_COUNT; i++) {
                GUARD(PARTICIPANT_PAGE_COUNT);
                createSmartObject((int32_t)&page,
sizeof(participant_page_t), (int32_t)&page.next_page_id);
            }

            lottery.first_participant_page_id = page.next_page_id;

            lottery.winning_value = 0x0000000000000000;
            lottery.winning_value_determined = FALSE;

            id256_t lottery_id;
            createSmartObject((int32_t)&lottery, sizeof(lottery_t),
(int32_t)&lottery_id);

            break;
        }
        case 2: // make commit
        {
            if (!paramsPassed()) return -21;

            params2_t params;
            // Read packed params with exact serialized size (no hidden
padding bytes).
            int32_t got = getParams((int32_t)&params,
sizeof(params2_t));
            if (got != sizeof(params2_t)) return -22;

            lottery_t lottery;
            int32_t sz = getSmartObjectData((int32_t)&params.lottery_id,
(int32_t)&lottery, sizeof(lottery_t));
            if (sz != sizeof(lottery_t)) return -23;

            if ((uint32_t)getLedgerTimestamp() > lottery.start_timestamp
+ COMMIT_WINDOW_DURATION) return -24;

            uint8_t commit_made = FALSE;
            id256_t page_id = lottery.first_participant_page_id;
```

```
            for (uint32_t i = 0; i < PARTICIPANT_PAGE_COUNT; i++) {
                GUARD(PARTICIPANT_PAGE_COUNT);

                participant_page_t page;
                getSmartObjectData((int32_t)&page_id, (int32_t)&page,
sizeof(participant_page_t));

                // 1 commit per participant
                for (uint32_t j = 0; j < page.filled_slots; j++) {
                    GUARD(PARTICIPANT_PAGE_COUNT *
PARTICIPANT_PAGE_CAPACITY);

                    if (addr_eq(&page.participants[j].address,
&contract_caller)) return -25;
                }

                if (page.filled_slots < PARTICIPANT_PAGE_CAPACITY) {
                    participant_t participant;
                    participant.address = contract_caller;
                    participant.revealed = FALSE;
                    // params2_t is packed, so commit bytes map 1:1 to
input serialization.
                    participant.entry = params.commit;
                    page.participants[page.filled_slots] = participant;
                    page.filled_slots++;
                    setSmartObject((int32_t)&page_id, (int32_t)&page,
sizeof(participant_page_t));
                    lockCallerXRP(ENTRY_DEPOSIT + ENTRY_FEE);
                    commit_made = TRUE;
                    break;
                }
                page_id = page.next_page_id;
            }
            if (!commit_made) return -26;

            break;
        }
        case 3: // reveal secret
        {
            if (!paramsPassed()) return -31;

            params3_t params;
```

```
            // Read packed params with exact serialized size (no hidden
padding bytes).
            int32_t got = getParams((int32_t)&params,
sizeof(params3_t));
            if (got != sizeof(params3_t)) return -32;

            lottery_t lottery;
            int32_t sz = getSmartObjectData((int32_t)&params.lottery_id,
(int32_t)&lottery, sizeof(lottery_t));
            if (sz != sizeof(lottery_t)) return -33;

            uint32_t ledger_timestamp = (uint32_t)getLedgerTimestamp();
            if (ledger_timestamp > lottery.start_timestamp +
COMMIT_WINDOW_DURATION + REVEAL_WINDOW_DURATION ||
                ledger_timestamp <= lottery.start_timestamp +
COMMIT_WINDOW_DURATION)
                return -34;

            uint8_t secret_revealed = FALSE;
            id256_t page_id = lottery.first_participant_page_id;
            for (uint32_t i = 0; i < PARTICIPANT_PAGE_COUNT; i++) {
                GUARD(PARTICIPANT_PAGE_COUNT);

                participant_page_t page;
                getSmartObjectData((int32_t)&page_id, (int32_t)&page,
sizeof(participant_page_t));

                for (uint32_t j = 0; j < page.filled_slots; j++) {
                    GUARD(PARTICIPANT_PAGE_COUNT *
PARTICIPANT_PAGE_CAPACITY);

                    if (addr_eq(&page.participants[j].address,
&contract_caller)) {
                        hash256_t commit_verif;
                        commit_unhashed_t commit_unhashed;
                        commit_unhashed.account_id = contract_caller;
                        commit_unhashed.contract_id = contract_id;
                        commit_unhashed.secret = params.secret;
                        // Hash packed preimage bytes to avoid hidden
padding drift.
                        sha256((int32_t)&commit_unhashed,
sizeof(commit_unhashed_t), (int32_t)&commit_verif);
```

```
                    if (!hash256_eq(&page.participants[j].entry,
&commit_verif))
                        return -35;
                    // Safe 64-bit write: participant_t.entry
alignment is asserted above.
                    page.participants[j].entry.words[0] =
params.secret;
                    page.participants[j].revealed = TRUE;
                    setSmartObject((int32_t)&page_id,
(int32_t)&page, sizeof(participant_page_t));
                    unlockXRP(ENTRY_DEPOSIT,
(int32_t)&page.participants[j].address);
                    secret_revealed = TRUE;
                    break;
                }
            }
            if (secret_revealed) break;
            page_id = page.next_page_id;
        }
        if (!secret_revealed) return -36;

        break;
    }
    case 4: // determine winning value
    {
        if (!paramsPassed()) return -41;

        id256_t lottery_id;
        int32_t got = getParams((int32_t)&lottery_id,
sizeof(id256_t));
        if (got != sizeof(id256_t)) return -42;

        lottery_t lottery;
        int32_t sz = getSmartObjectData((int32_t)&lottery_id,
(int32_t)&lottery, sizeof(lottery_t));
        if (sz != sizeof(lottery_t)) return -43;

        if ((uint32_t)getLedgerTimestamp() < lottery.start_timestamp
+ COMMIT_WINDOW_DURATION + REVEAL_WINDOW_DURATION)
            return -44;

        uint64_t winning_value = 0x0000000000000000;
```

```
            id256_t page_id = lottery.first_participant_page_id;
            for (uint32_t i = 0; i < PARTICIPANT_PAGE_COUNT; i++) {
                GUARD(PARTICIPANT_PAGE_COUNT);

                participant_page_t page;
                getSmartObjectData((int32_t)&page_id, (int32_t)&page,
sizeof(participant_page_t));

                for (uint32_t j = 0; j < page.filled_slots; j++) {
                    GUARD(PARTICIPANT_PAGE_COUNT *
PARTICIPANT_PAGE_CAPACITY);

                    if (page.participants[j].revealed)
                        // Safe 64-bit read: participant_t.entry
alignment is asserted above.
                        winning_value ^=
page.participants[j].entry.words[0];
                }
                page_id = page.next_page_id;
            }

            lottery.winning_value = winning_value;
            lottery.winning_value_determined = TRUE;
            setSmartObject((int32_t)&lottery_id, (int32_t)&lottery,
sizeof(lottery_t));

            break;
        }
        case 5: // calculate & pay winner
        {
            if (!paramsPassed()) return -51;

            id256_t lottery_id;
            int32_t got = getParams((int32_t)&lottery_id,
sizeof(id256_t));
            if (got != sizeof(id256_t)) return -52;

            lottery_t lottery;
            int32_t sz = getSmartObjectData((int32_t)&lottery_id,
(int32_t)&lottery, sizeof(lottery_t));
            if (sz != sizeof(lottery_t)) return -53;
```

```
            if ((uint32_t)getLedgerTimestamp() < lottery.start_timestamp
+ COMMIT_WINDOW_DURATION + REVEAL_WINDOW_DURATION)
                return -54;

            if (!lottery.winning_value_determined) return -55;

            uint64_t lowest_difference = 0xFFFFFFFFFFFFFFFF;
            addr_t winner;

            id256_t page_id = lottery.first_participant_page_id;
            for (uint32_t i = 0; i < PARTICIPANT_PAGE_COUNT; i++) {
                GUARD(PARTICIPANT_PAGE_COUNT);

                participant_page_t page;
                getSmartObjectData((int32_t)&page_id, (int32_t)&page,
sizeof(participant_page_t));

                for (uint32_t j = 0; j < page.filled_slots; j++) {
                    GUARD(PARTICIPANT_PAGE_COUNT *
PARTICIPANT_PAGE_CAPACITY);

                    if (page.participants[j].revealed) {
                        // Safe 64-bit read: participant_t.entry
alignment is asserted above.
                        uint64_t difference =
page.participants[j].entry.words[0] ^ lottery.winning_value;
                        if (difference < lowest_difference) {
                            winner = page.participants[j].address;
                            lowest_difference = difference;
                        }
                    }
                }

                deleteSmartObject((int32_t)&page_id);
                page_id = page.next_page_id;
            }

            deleteSmartObject((int32_t)&lottery_id);

            if (lowest_difference < 0xFFFFFFFFFFFFFFFF)
                unlockXRP(PRIZE, (int32_t)&winner);

            int64_t contract_balance = getContractBalance();
```

```
            if (contract_balance > 0)
                unlockXRP(contract_balance, (int32_t)&contract_owner);

            break;
        }
        default:
        {
            return -999;
        }
    }

    return 0;
}
```