SMART CONTRACT FORKS:
MIGRATING BETWEEN BLOCKCHAIN NETWORKS

Martin Westerkamp
Service-centric Networking | TU Berlin & Telekom Innovation Laboratories
BACKGROUND
SMART CONTRACTS - PROBLEM STATEMENT

Dependency on host blockchain
• Lock-in effect
• Inherits host blockchain’s properties
  • Security
  • Vulnerabilities of blockchain
  • Performance
• Incentives of smart contract users are detached from those of miners

External Requirements
• Requirements may change during life-cycle
• Better suited blockchain implementations may be available over time
• Host blockchain performance and costs may change
IDEAS & BASICS
SMART CONTRACT FORKS

Smart Contract Logic
• Retrieve execution logic (bytecode) from source blockchain
• A common execution environment is required

Smart Contract State
• Changes over time
• Should be forkable at any point in time
• The source contract should not be affected

Verifiability
• Migrations should not require trust in the executing entity
• Logic and state should be verifiably equivalent between source and target blockchain
IDEAS & BASICS
EVM-COMPATIBLE BLOCKCHAIN IMPLEMENTATIONS

Blockchain Variations
• Permissioned vs. Permissionless
• Public vs. Private

Blockchain implementations
• A multitude of implementations exists
• Blockchains cannot query external information
• Blockchains cannot access the state of other blockchains

Dimensions
• Consensus protocol
  • Proof-of-Work
  • (delegated) Proof-of-Stake
  • Practical Byzantine Fault Tolerance (PBFT)
IDEAS & BASICS
EVM-BASED SMART CONTRACTS

Smart Contract Programming
- Smart contracts are developed in high level languages, e.g. Solidity or Vyper
- High level code is compiled to bytecode
- Bytecode contains instructions to create an account and copy contract code

Bytecode
- Instructions interpreted by a virtual machine
- Stack-based environment
IDEAS & BASICS
ETHEREUM BLOCKCHAIN

Merkle trees
- Root hashes of three distinct Merkle trees
  - Transactions contained in the block
  - State root representing the current global state
  - Receipt tree containing receipts for executed transactions

State representation
- Every user and smart contract maintains a dedicated account
- Every smart contract maintains a dedicated Merkle tree
- Merkle-proofs allow efficient verifiability of included data

Challenge
- Storage keys are not (necessarily) sequential
  - Mappings use a concatenation with their key
  - Keys are hashed as a protection against DoS attacks building deep trees
SMART CONTRACT FORKS
GENERAL STATE RETRIEVAL AND CONTRACT RECONSTRUCTION

Alice
1. deploy contract
Deploy code
a = 2

Bob
2. a := 5

replay transactions

Contract code:
0x6080...
State:
1. Replay all TXs
2. Create key/value map

Carol
3. extract contract

4. Chain B (target)

Deploy code
a = 2

Contract code:
0x6080...
State:
1. Replay all TXs
2. Create key/value map

Dave
6. deploy contract
get state root
State root: 0x1337
7. get (a)
a = 5

State root: 0x1337

SMART CONTRACT FORKS: MIGRATING BETWEEN BLOCKCHAIN NETWORKS | MARTIN WESTERKAMP
SMART CONTRACT FORKS
STATE RETRIEVAL AND CONTRACT RECONSTRUCTION W/ FATDB

1. **Deploy contract**
   - **Deploy code**
     - **a = 2**

2. **Extract contract**
   - **Contract code:** 0x6080...
   - **State:**
     1. Replay all TXs
     2. Create key/value map

3. **Alice**
   - **a := 5**

4. **Bob**
   - **Carol**
     - **Create bytecode**

5. **Create bytecode**
   - **State root:** 0x1337

6. **Deploy contract**
   - **address:** 0x42...
   - **a := 5**

7. **Deploy contract**
   - **address:** 0x73...
   - **State root:** 0x1337
   - **Dave**
     - **Get state root**
     - **Get a**
     - **a = 5**

8. **Get state root**
   - **State root:** 0x1337
SMART CONTRACT FORKS
LARGE CONTRACT STATES

Challenge

• Large contract code size and large states may exceed block gas limit for deploying in a single transaction
• Adding a setter function alters the contract and decreases verifiability
  • Contract code would be modified
  • An additional variable tracking setting phase would be required to prevent future modifications -> different storage root

Solution

• Target Contract is split in three smart contracts:
  • Logic Contract
  • Proxy Contract
  • Initialization Contract
SMART CONTRACT FORKS
DEPENDENCIES

Smart Contracts Dependencies

- Inheritance (hard coded)
- References
  - Hard coded, e.g. libraries
  - Variable, maintained in state

Resolving Dependencies

- Analyze
  - bytecode
  - state variables
- Replace references with new addresses in state and bytecode
EVALUATION
ERC20 CONTRACTS

Example Use-Case

- ERC20 token contracts are one of the most used contract types
- A random ERC20 contract has been chosen from Etherscan on the mainnet
  - 3,179 transactions
  - State observed at block height 6,837,908
- Target blockchain is a private Ethereum instance

Migration

- External transaction database used for observing relevant transactions (internal and external TXs)
- Final state consists of 759 key/value pairs
- Size requires splitting the target contract
  - 20,000gas * 759 = 15,180,000gas
  - Gas limit: 8,000,000

Additional gas costs

- Splitting into logic and state contract requires delegate calls for each function call
  - 956gas additional gas is required
- Single contract deployment is preferred

Vulnerability

- The logic contract could be addressed directly changing its state
- In case a selfdestruct instruction is exposed, it could be unprotected, leading to a potential vulnerability
- Contracts need to be analyzed for such instructions in advance
Example Use-Case

- Contract containing hard-coded and variable references
- Target: Permissioned Quorum instance

Migration

- Referenced contracts are recognized
- Respective contracts are migrated recursively
- Addresses are updated in state and bytecode

Verifiability

- A map of replaced values is required for validation
- For states, it is not sufficient to compare state roots in this case
Blockchain Forks
- Most blockchains do not guarantee finality
- Once validated contract migrations may become invalid
- Any user can decide when to assume finality

Address patterns
- Source and target ledger require equal address patterns, so that the same key pair can be used on the target ledger to claim ownership of tokens etc.
- Distinct patterns may require a redeeming phase
  - E.g., a contract on the source ledger
  - Comparable to ICOs as EOS

Proofs using chain relays
- Validating blockchain header from the source chain on the target ledger permit proving equality on-chain
- Potentially permits contract synchronization

Native cryptocurrency
- Smart contracts may hold native coins like Ether
  - Not part of the contract's storage trie
  - Stored as account value
- Smart contract logic may depend on stored cryptocurrency
- Notary networks like Cosmos or Polkadot may be utilized in the future

Events
- Events emitted are not directly linked to accounts, but in a dedicated Merkle tree within each block
- Some applications may require access to historical events
- A middleware could direct such requests in the future
CONCLUSION

Smart Contract Forks
- Smart Contract can be migrated between EVM-compatible Blockchains
  - Contract Logic
  - State

Trust
- Any participant can verify the migration’s validity
- No trust in migrating entity is required

Flexibility
- Depending on the source contract, different migration techniques are applied

Tooling
- We supply an open source tool that supports migrations and validation
  - https://github.com/informartin/VeriSmart

https://unsplash.com/photos/bj3l739cw8
Martin Westerkamp
TU Berlin | Telekom Innovation Laboratories | Fachgebiet Service-centric Networking
Ernst-Reuter-Platz 7 | 10587 Berlin | Germany
westerkamp@tu-berlin.de
https://www.snet.tu-berlin.de/westerkamp