Ethereum Virtual Machine and Execution Environment Overview

Before Ethereum

- Specialized blockchains/protocols for specialized applications
  - Bitcoin
  - Namecoin
  - Colored coins
- “Swiss army knife” protocols
- Problem: insufficiently general
  - How can a protocol be future-compatible with applications which have not been invented yet and which we know nothing about?

Ethereum

- Core idea: instead of explicitly supporting applications, support a native programming language
- Business logic for applications can be implemented in this language
- Transactions
  - Bitcoin: send X bitcoins from A to B
  - Namecoin: register domain X, set IP address of domain X to Y (simplified)
  - Ethereum: call function F of contract C with arguments A (roughly)

Ethereum Account Model

- Two types of accounts
  - “Externally owned account” (controlled by private key)
  - Contract (controlled by its own code)
  - Note: this will change in future hard forks as we move toward a one-account-type model where private-key-controlled accounts are themselves implemented via contracts
- Accounts have a 20-byte address
- Contracts also have:
  - Code (static)
  - Storage (key/value store, can only be read/written by parent account)

Ethereum Account Model

- A transaction contains the following key parameters:
  - Gas limit
  - Destination address
  - Data
  - Sequence number + signature
- If the destination address has code, the code runs
- Code has the ability to:
  - Read/write to storage
  - Send “internal transactions” to other accounts
    - These transactions give the parent execution instance a return value, hence they can be viewed as function calls

Ethereum Account Model

- Accounts can be used for multiple functions
  - Specify an access policy for an individual or organization (“eg. any of these 5 keys can spend up to 100 coins per day, but if 3 out of 5 approve an operation then it can spend any amount”)
  - Maintain a database of who owns how much of an asset and process “send” operations
  - Specify an agreement between multiple parties to split the funds between them based on some conditions (eg. financial derivatives)
  - Escrow
  - Store data that can be queried by other contracts (eg. Info about which accounts are authorized users for some system)

Ethereum Virtual Machine

- Inputs: code, data, environment variables (eg. block number, timestamp)
- Externs: read/write storage, make sub-calls (aka internal transactions), make logs, etc
- Theoretically, many different virtual machines can be modified and “hooked up” to this interface, so you can use the EAM without the EVM

EVM Requirements

- Small code size (so that very many contracts from many users can be stored by one node)
- VM security designed around running untrusted code from arbitrary parties
- Multiple implementations (for cross-checking, and to mitigate developer centralization in the public chain)
- Perfect determinism (for consensus)
- Infinite loop resistance
  - This itself must be accomplished perfectly deterministically; timeouts are a no-go

Metering/Gas

- Approximately: count computational steps
- If a transaction or sub-call “runs out of gas”, it is fully reverted (to preserve atomicity), though the gas is still treated as fully consumed (to prevent DoS attacks)
- Many operations have different gas costs; this incorporates:
  - Runtime
  - Consumption of memory and storage
  - Consumption of bandwidth
  - “Pollution” of the log bloom filter

Pre-compiles

- Some cryptographic operations are currently too slow to be done on EVM
- Hence, we provide native versions that can be used by calling accounts at pre-specified addresses
- 1 = elliptic curve signature verification (technically, public key hash recovery)
- 2 = SHA256
- 3 = RIPEMD160
- 4 = identity (for efficient data copying)

ABI

- Most contracts have multiple functions that you may want to call
- Idea: first 4 bytes of transaction/call data are the function identifier, remaining data represents arguments in a standardized format (each argument padded to 32 bytes to increase efficiency of VM processing)

High-level languages

- Developers do not want to program in raw EVM assembly
- Write code in higher-level language, compiler compiles down to EVM code
- HLLs include: Serpent, LLL, Solidity (most popular)
- Currently in progress: formal verification on Solidity
- Future ideas: total functional HLLs, DSLs (eg. a financial DSL focusing on flows as the fundamental unit)

Integration opportunities for Hyperledger

- Keep existing account model, offer EVM as an option for computation
  - Contract storage?
  - Internal transactions / calls?
- Integrate both account model and EVM
- Offer EVM, work together on account model to have desired properties for both smart contracts and other use cases?

Future development

- Merge two account types
  - Contract code itself would specify account “access policy” for all accounts
- Sharding
  - Allows scalability through parallelizability, through restricting transaction effects to particular address ranges
  - “Cross-shard” operations must be done asynchronously
- Increase efficiency for implementing cryptography on top of EVM
  - Additional precompiles (eg. ECMUL/ECADD for ring signatures, bigint math for RSA)
  - Experimental architecture based on a metering trans-compiler for WASM code