Ethereum: mechanics

Dan Boneh

Note: HW#2 is posted on the course web site. Due Oct. 18.
Limitations of Bitcoin

Recall: UTXO contains (hash of) ScriptPK
• simple script: indicates conditions when UTXO can be spent

Limitations:
• Difficult to maintain state in multi-stage contracts
• Difficult to enforce global rules on assets

A simple example: rate limiting. My wallet manages 100 UTXOs.
• Desired policy: can only transfer 2BTC per day out of my wallet
An example: NameCoin

Domain name system on the blockchain: [google.com → IP addr]

Need support for three operations:

• **Name.new**(OwnerAddr, DomainName): intent to register
• **Name.update**(DomainName, newVal, newOwner, OwnerSig)
• **Name.lookup**(DomainName)

Note: also need to ensure no front-running on **Name.new**()
A broken implementation

Name.new() and Name.update() create a UTXO with ScriptPK:

```
DUP HASH256 <OwnerAddr> EQVERIFY CHECKSIG VERIFY
<NAMESCOIN> <DomainName> <IPaddr> <1>
```

only owner can “spend” this UTXO to update domain data

**Contract:** (should be enforced by miners)

- if domain google.com is registered,
- no one else can register that domain

Problem: this contract cannot be enforced using Bitcoin script

verify sig is valid

ensure top of stack is 1
What to do?

NameCoin: fork of Bitcoin that implements this contract
(see also the Handshake, Chia projects)

Can we build a blockchain that natively supports generic contracts like this?

⇒ Ethereum
Ethereum: enables a world of applications

A world of Ethereum Decentralized apps (DAPPs)

• New coins: ERC-20 interface to DAPP
• DeFi: exchanges, lending, stablecoins, derivatives, etc.
• Insurance
• DAOs: decentralized organizations
• NFTs: Managing distinguished assets (ERC-721 interface)
• Games, metaverse: assets managed on chain

stateofthedapps.com, dapp.review
Bitcoin as a state transition system

world state

UTXO₁
UTXO₂
⋮

updated world state

UTXO₁
UTXO₃
⋮

input

Tx: UTXO₂ → UTXO₃

Bitcoin rules:

\[ F_{\text{bitcoin}} : S \times I \rightarrow S \]

S: set of all possible world states, \( s_0 \in S \) genesis state
I: set of all possible inputs
Ethereum as a state transition system

Much richer state transition functions

⇒ one transition executes an entire program
Running a program on a blockchain (DAPP)

compute layer (executed by miners): The EVM

consensus layer

create a DAPP

program code

state₀ → Tx₁ → state₁ → Tx₂ → state₂ → ...

... blockchain ...

Running a program on a blockchain (DAPP)
The Ethereum system

Layer 1 (ETHv1): PoW consensus. Block reward = 2 ETH + Tx fees (gas)

avg. block rate = 15 seconds.

ETHv1: variant of Nakamoto PoW

ETHv2: proof of stake consensus

about 150 Tx per block.
World state: set of accounts identified by 32-byte address.

Two types of accounts:

(1) **owned accounts**: controlled by ECDSA signing key pair (PK,SK).
   SK signing key known only to account owner

(2) **contracts**: controlled by code.
   code set at account creation time, does not change
## Data associated with an account

<table>
<thead>
<tr>
<th>Account data</th>
<th>Owned</th>
<th>Contracts</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>address</strong> <em>(computed):</em></td>
<td>( H(PK) )</td>
<td>( H(\text{CreatorAddr}, \text{CreatorNonce}) )</td>
</tr>
<tr>
<td><strong>code:</strong></td>
<td>( \perp )</td>
<td>CodeHash</td>
</tr>
<tr>
<td><strong>storage root</strong> <em>(state):</em></td>
<td>( \perp )</td>
<td>StorageRoot</td>
</tr>
<tr>
<td><strong>balance</strong> <em>(in Wei):</em></td>
<td>balance</td>
<td>balance ((10^{18} \text{ Wei} = 1 \text{ ETH}))</td>
</tr>
<tr>
<td><strong>nonce:</strong></td>
<td>nonce</td>
<td>nonce</td>
</tr>
</tbody>
</table>

\((\#\text{Tx sent}) + (\#\text{accounts created})\): anti-replay mechanism
Account state: persistent storage

Every contract has an associated **storage array $S[]$**:

$S[0], S[1], \ldots, S[2^{256}-1]$: each cell holds 32 bytes, init to 0.

Account storage root: **Merkle Patricia Tree hash of $S[]$**

- Cannot compute full Merkle tree hash: $2^{256}$ leaves

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$a$</td>
<td>$b$</td>
<td>$c$</td>
<td>$d$</td>
</tr>
</tbody>
</table>

![Merkle Patricia Tree Diagram]

- Time to compute root hash: $\leq 2 \times |S|$
- $|S| = \# \text{non-zero cells}$
Transactions: signed data by initiator

- **To:** 32-byte address of target (0 → create new account)
- **From, [Signature]:** initiator address and signature on Tx (if owned)
- **Value:** # Wei being sent with Tx
- **Tx fees (EIP 1559):** `gasLimit`, `maxFee`, `maxPriorityFee` (later)
- if **To = 0:** create new contract `code = (init, body)`
- if **To ≠ 0:** `data` (what function to call & arguments)
- **nonce:** must match current nonce of sender (prevents Tx replay)
Transaction types:

owned $\rightarrow$ owned: transfer ETH between users

owned $\rightarrow$ contract: call contract with ETH & data
<table>
<thead>
<tr>
<th>From</th>
<th>To</th>
<th>msg.value</th>
<th>Tx fee (ETH)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xa4ec1125ce9428ae5...</td>
<td>0x2cebe81fe0dcd220e...</td>
<td>0 Ether</td>
<td>0.00404405</td>
</tr>
<tr>
<td>0xba272f30459a119b2...</td>
<td>Uniswap V2: Router 2</td>
<td>0.14 Ether</td>
<td>0.00644563</td>
</tr>
<tr>
<td>0x4299d864bbda0fe32...</td>
<td>Uniswap V2: Router 2</td>
<td>89,389104111882671 Ether</td>
<td>0.00716578</td>
</tr>
<tr>
<td>0x4d1317a2a98cfea41...</td>
<td>0xc59f33af5f4a7c8647...</td>
<td>14.501 Ether</td>
<td>0.001239</td>
</tr>
<tr>
<td>0x29ecaa773f052d14e...</td>
<td>CryptoKitties: Core</td>
<td>0 Ether</td>
<td>0.00775543</td>
</tr>
<tr>
<td>0x63bb46461696416fa...</td>
<td>Uniswap V2: Router 2</td>
<td>0.203036474328481 Ether</td>
<td>0.00766728</td>
</tr>
<tr>
<td>0xde70238ae7a35abd...</td>
<td>Balancer: ETH/DOUGH...</td>
<td>0 Ether</td>
<td>0.00261582</td>
</tr>
<tr>
<td>0x69aca10fe1394d535f...</td>
<td>0x837d03aa7f09b8be...</td>
<td>0 Ether</td>
<td>0.00259936</td>
</tr>
<tr>
<td>0xe2f5d180626d29e75...</td>
<td>Uniswap V2: Router 2</td>
<td>0 Ether</td>
<td>0.00665809</td>
</tr>
</tbody>
</table>
Messages: virtual Tx initiated by a contract

Same as Tx, but no signature  (contract has no signing key)

contract → owned:  contract sends funds to user
contract → contract:  one program calls another (and sends funds)

One Tx from user: can lead to many Tx processed.  Composability!

Tx from owned addr → contract → another contract

another contract → different owned
Example Tx

**State**

- `14c5f8ba: 1024 eth`
- `bb75a980: 5202 eth`
  - `contract`
  - `tx.data[0] = tx.data[1]`
  - `[0, 235235, 0, ALICE ....]`
- `892bf92f: 0 eth`
  - `contract`
  - `tx.value / 3, contract.storage[0]`
  - `tx.value / 3, contract.storage[1]`
  - `tx.value / 3, contract.storage[2]`
  - `[ALICE, BOB, CHARLIE ]`
- `4096ad65: 77 eth`

**State'**

- `14c5f8ba: 1014 eth`
- `bb75a980: 5212 eth`
  - `contract`
  - `tx.data[0] = tx.data[1]`
  - `[0, 235235, CHARLIE, ALICE ..]`
- `892bf92f: 0 eth`
  - `contract`
  - `tx.value / 3, contract.storage[0]`
  - `tx.value / 3, contract.storage[1]`
  - `tx.value / 3, contract.storage[2]`
  - `[ALICE, BOB, CHARLIE ]`
- `4096ad65: 77 eth`

**Transaction**

- **From:** `14c5f8ba`
- **To:** `bb75a980`
- **Value:** `10 eth`
- **Data:** `2, CHARLIE`
- **Sig:** `30452fde3db3f7959f2ce8a1`

**world state (four accounts)**

**updated world state**
Miners collect Txs from users \( \Rightarrow \) leader creates a block of \( n \) Tx

- Miner does:
  - for \( i=1,\ldots,n \): execute state change of \( \text{Tx}_i \) sequentially
    (can change state of \( >n \) accounts)
  - record updated world state in block

Other miners re-execute all Tx to verify block

- Miners should only build on a valid block
- Miners are not paid for verifying block (note: verifier’s dilemma)
Block header data (simplified)

1. consensus data: parent hash, difficulty, PoW solution, etc.

2. address of gas beneficiary: where Tx fees will go

3. **world state root**: updated world state
   
   Merkle Patricia Tree hash of **all** accounts in the system

4. **Tx root**: Merkle hash of all Tx processed in block

5. **Tx receipt root**: Merkle hash of log messages generated in block

5. Gas used: tells verifier how much work to verify block
The Ethereum blockchain: abstractly
Amount of memory to run a node (in GB)

≈1 TB

ETH total blockchain size: 8.6 TB (Oct. 2021)
contract nameCoin {  // Solidity code (next lecture)

  struct nameEntry {
    address owner;  // address of domain owner
    bytes32 value;  // IP address
  }

  // array of all registered domains
  mapping (bytes32 => nameEntry) data;
function **nameNew**(bytes32 name) {

    // registration costs is 100 Wei

    if (data[name] == 0 && msg.value >= 100) {
        data[name].owner = msg.sender  // record domain owner
        emit Register(msg.sender, name)  // log event
    }
}

Code ensures that no one can take over a registered name

Serious bug in this code! Front running. Solved using commitments.
function nameUpdate(
    bytes32 name, bytes32 newValue, address newOwner) {

    // check if message is from domain owner,
    // and update cost of 10 Wei is paid

    if (data[name].owner == msg.sender && msg.value >= 10) {
        data[name].value = newValue;       // record new value
        data[name].owner = newOwner;       // record new owner
    }
}
An example contract: NameCoin

function nameLookup(bytes32 name) {
    return data[name];
}

} // end of contract
Write code in Solidity (or another front-end language)

⇒ compile to EVM bytecode

   (some projects use WASM or BPF bytecode)

⇒ miners use the EVM to execute contract bytecode in response to a Tx
The EVM

Stack machine (like Bitcoin) but with JUMP
• max stack depth = 1024
• program aborts if stack size exceeded; miner keeps gas
• contract can create or call another contract

In addition: two types of zero initialized memory
• Persistent storage (on blockchain): SLOAD, SSTORE (expensive)
• Volatile memory (for single Tx): MLOAD, MSTORE (cheap)
• LOG0(data): write data to log

see https://ethervm.io/
Every instruction costs gas, examples:

**SSTORE** `addr` (32 bytes), `value` (32 bytes)

- zero → non-zero: 20,000 gas
- non-zero → non-zero: 5,000 gas
- non-zero → zero: 15,000 gas refund

Refund is given for reducing size of blockchain state

**SELFDESTRUCT** `addr`: kill current contract. 24,000 gas refund

**CREATE**: 32,000 gas  **CALL** `gas`, `addr`, `value`, `args`
Why charge gas?

- Tx fees (gas) prevents submitting Tx that runs for many steps.
- During high load: miners choose Tx from the mempool that maximize their income.

Old EVM: (prior to EIP1559, live on 8/2021)

- Every Tx contains a gasPrice “bid” (gas $\rightarrow$ Wei conversion price)
- Miners choose Tx with highest gasPrice ($\max \sum (\text{gasPrice} \times \text{gasLimit})$)

$\Rightarrow$ not an efficient auction mechanism (first price auction)
Gas prices spike during congestion

GasPrice in Gwei:

$$86 \text{ Gwei} = 86 \times 10^{-9} \text{ ETH}$$

Average Tx fee in USD
Every block has a “baseFee”:

- the minimum gasPrice for all Tx in the block

- baseFee is computed from total gas in earlier blocks:
  - earlier blocks at gas limit (30M gas) \(\Rightarrow\) base fee goes up 12.5%
  - earlier blocks empty \(\Rightarrow\) base fee decreases by 12.5%

If earlier blocks at “target size” (15M gas) \(\Rightarrow\) base fee does not change
Gas calculation

EIP1559 Tx specifies three parameters:

- **gasLimit**: max total gas allowed for Tx
- **maxFee**: maximum allowed gas price (max gas \( \rightarrow \) Wei conversion)
- **maxPriorityFee**: additional “tip” to be paid to miner

Computed **gasPrice** bid:

\[
\text{gasPrice} \leftarrow \min(\text{maxFee}, \quad \text{baseFee} + \text{maxPriorityFee})
\]

Max Tx fee: \( \text{gasLimit} \times \text{gasPrice} \)
Gas calculation

(1) if gasPrice < baseFee: abort
(2) If gasLimit × gasPrice < msg.sender.balance: abort
(3) deduct gasLimit × gasPrice from msg.sender.balance

(4) set Gas ← gasLimit
(5) execute Tx: deduct gas from Gas for each instruction
    if at end (Gas < 0): abort, Tx is invalid (miner keeps gasLimit × gasPrice)
(6) Refund Gas × gasPrice to msg.sender.balance

(7) gasUsed ← gasLimit – Gas
    (7a) BURN gasUsed × baseFee
    (7b) Send gasUsed × (gasPrice – baseFee) to miner
Burn results in practice

block reward (2ETH) –  
Total baseFee burned in block

... sometimes burn exceeds block rewards  \( \implies \) ETH deflation
EIP1559 goals (informal):

• users incentivized to bid their true utility for posting Tx,
• miners incentivized to not create fake Tx, and
• disincentivize off chain agreements.

Suppose no burn (i.e., baseFee given to miners):

⇒ in periods of low Tx volume miners would try to increase volume by offering to refund the baseFee off chain to users.
Note: transactions are becoming more complex

Total Gas Usage

Evolution of the total gas used by the Ethereum network per day

Gas usage is increasing  ⇒  each Tx takes more instructions to execute
Next lecture: writing Solidity contracts