HW#1 is posted on the course web site. Due Oct. 4
(1) **SHA256**: a collision resistant hash function that outputs 32-byte hash values

**Applications:**

- a binding commitment to one value: \( \text{commit}(m) \rightarrow H(m) \)
  or to a list of values: \( \text{commit}(m_1, \ldots, m_n) \rightarrow \text{Merkle}(m_1, \ldots, m_n) \)

- Proof of work with difficulty \( D \):
  given \( x \) find \( y \) s.t. \( H(x, y) < 2^{256}/D \) takes time \( O(D) \)
(2) Digital signatures: \((\text{Gen, Sign, Verify})\)

\[\text{Gen}() \rightarrow (pk, sk),\]

\[\text{Sign}(sk, m) \rightarrow \sigma, \quad \text{Verify}(pk, m, \sigma) \rightarrow \text{accept/reject}\]

signing key

verification key
This lecture: Bitcoin mechanics

- Jan. 2009: Bitcoin network launched
- Sep. 2021: $800B

Total market value:
This lecture: Bitcoin mechanics

- **user facing tools** (cloud servers)
- **applications** (DAPPs, smart contracts)
- **compute layer** (blockchain computer)
- **consensus layer**

next week
First: overview of the Bitcoin consensus layer

Bitcoin P2P network

end users

signed Tx

sk_A

sk_B

sk_C

typically, miners are connected to eight other peers (anyone can join)
First: overview of the Bitcoin consensus layer

miners broadcast received Tx to the P2P network

every miner:
  validates received Tx and stores them in its mempool (unconfirmed Tx)

note: miners see all Tx before they are posted on chain
First: overview of the Bitcoin consensus layer

Every 10 minutes:

- Each miner creates a candidate block from Tx in its mempool
- a “random” miner is selected (how: next week), and broadcasts its block to P2P network
- all miners validate new block

I am the leader

blockchain
Selected miner is paid 6.25 BTC in **coinbase Tx** (first Tx in the block)

• only way new BTC is created
• block reward halves every four years
  ⇒ max 21M BTC (currently 18.75M BTC)

note: miner chooses order of Tx in block
Properties (very informal)

Next week:

**Persistence:**
- to remove a block, need to convince 51% of mining power *

**Liveness:**
- to block a Tx from being posted, need to convince 51% of mining power **
  (some sub 50% censorship attacks, such as feather forks)
Bitcoin blockchain: a sequence of block headers, 80 bytes each

- **genesis block**
- **BH$_1$**
  - version (4 bytes)
  - prev (32 bytes)
  - time (4 bytes)
  - bits (4 bytes)
  - nonce (4 bytes)
  - Tx root (32 bytes)

80 bytes

- **BH$_2$**
  - prev
  - Tx root

- **BH$_3$**
  - prev
  - Tx root

...
**Bitcoin blockchain:** a sequence of block headers, 80 bytes each

**time:** time miner assembled the block. Self reported. (block rejected if too far in past or future)

**bits:** proof of work difficulty

**nonce:** proof of work solution

**Merkle tree:** payer can give a short proof that Tx is in the block

new block every $\approx 10$ minutes.
An example (Sep. 2020)

<table>
<thead>
<tr>
<th>Height</th>
<th>Mined</th>
<th>Miner</th>
<th>Size</th>
<th>#Tx</th>
</tr>
</thead>
<tbody>
<tr>
<td>648494</td>
<td>17 minutes</td>
<td>Unknown</td>
<td>1,308,663 bytes</td>
<td>1855</td>
</tr>
<tr>
<td>648493</td>
<td>20 minutes</td>
<td>SlushPool</td>
<td>1,317,436 bytes</td>
<td>2826</td>
</tr>
<tr>
<td>648492</td>
<td>59 minutes</td>
<td>Unknown</td>
<td>1,186,609 bytes</td>
<td>1128</td>
</tr>
<tr>
<td>648491</td>
<td>1 hour</td>
<td>Unknown</td>
<td>1,310,554 bytes</td>
<td>2774</td>
</tr>
<tr>
<td>648490</td>
<td>1 hour</td>
<td>Unknown</td>
<td>1,145,491 bytes</td>
<td>2075</td>
</tr>
<tr>
<td>648489</td>
<td>1 hour</td>
<td>Poolin</td>
<td>1,359,224 bytes</td>
<td>2622</td>
</tr>
</tbody>
</table>
### Block 648493

<table>
<thead>
<tr>
<th>Timestamp</th>
<th>2020-09-15 17:25</th>
</tr>
</thead>
<tbody>
<tr>
<td>Height</td>
<td>648493</td>
</tr>
<tr>
<td>Miner</td>
<td>SlushPool</td>
</tr>
</tbody>
</table>

(from coinbase Tx)

(adjusts every two weeks)

<table>
<thead>
<tr>
<th>Number of Transactions</th>
<th>2,826</th>
</tr>
</thead>
<tbody>
<tr>
<td>Difficulty</td>
<td>(D)</td>
</tr>
<tr>
<td></td>
<td>17,345,997,805,929.09</td>
</tr>
<tr>
<td>Merkle root</td>
<td>350cbb917c918774c93e945b960a2b3ac1c8d448c2e67839223bbcf595baff89</td>
</tr>
<tr>
<td>Transaction Volume</td>
<td>11256.14250596 BTC</td>
</tr>
<tr>
<td>Block Reward</td>
<td>6.25000000 BTC</td>
</tr>
<tr>
<td>Fee Reward</td>
<td>0.89047154 BTC</td>
</tr>
</tbody>
</table>

(Tx fees given to miner in coinbase Tx)
This lecture

View the blockchain as a sequence of Tx  (append-only)

coinbase Tx
Tx structure (non-coinbase)

inputs
- input[0]
- input[1]
- input[2]

outputs
- output[0]
- output[1]

(segwit)
- witnesses (part of input)

(4 bytes)
- locktime

TxID = H(Tx) (excluding witnesses)

input:
- TxID
- out-index
- ScriptSig
- seq

output:
- value
- ScriptPK

#BTC = value/10^8

32 byte hash
4 byte index
program
ignore
8 bytes
program

earliest block # that can include Tx
Example

Tx1: (funding Tx)

UTXO: unspent Tx output

Tx2: (spending Tx)

identifies a UTXO
Example

Tx1: (funding Tx)

UTXO: unspent Tx output

Tx2: (spending Tx)

identifies a UTXO

null locktime
Validating Tx2

Miners check (for each input):

1. The program \( \text{ScriptSig} \mid \text{ScriptPK} \) returns true

2. \( \text{TxID} \mid \text{index} \) is in the current UTXO set

3. sum input values \( \geq \) sum output values

After Tx2 is posted, miners remove UTXO_2 from UTXO set
### An example (block 648493) [2826 Tx]

<table>
<thead>
<tr>
<th>Input Address</th>
<th>Output Address</th>
<th>Tx Value (BTC)</th>
<th>Value (BTC)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3PuJbxJS1pKxf8EdVR18yBkD1fPAbgUtyw</td>
<td>1E5Ao1VUnA5BhffvXf2Xmud6avUgwkFnJv</td>
<td>0.72333974 BTC</td>
<td>0.072328174 BTC</td>
</tr>
<tr>
<td>17MWze4Z1uPjnvqj7SAnGtxcoVq11H8A</td>
<td>3G3C2RFQ8gsf77EQpdR4ZReChWFKEHhxVU</td>
<td>0.05000000 BTC</td>
<td>0.04808000 BTC</td>
</tr>
</tbody>
</table>

**Sum of fees in block added to coinbase Tx**

**COINBASE (Newly Generated Coins)**

<table>
<thead>
<tr>
<th>Address</th>
<th>Value (BTC)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1CK6KHY6MHgYvmRQ4PAafKYDrg1ejbH1cE</td>
<td>7.14047154 BTC</td>
</tr>
</tbody>
</table>

- **Tx0**: 0.00000000 BTC
- **Tx1**: 0.00005800 BTC
- **Tx2**: 0.00192000 BTC
- **Op_Return**: 0.00000000 BTC
- **Op_Return**: 0.00000000 BTC

6.25 + Tx fees = 7.14047154 BTC
Tx fees

Bitcoin average Tx fees in USD

Ethereum average Tx fees in USD
All value in Bitcoin is held in UTXOs

Unspent Transaction Outputs

The total number of valid unspent transactions outputs. This excludes invalid UTXOs with opcode OP_RETURN

Sep. 2021: miners need to store ≈76M UTXOs in memory
Focusing on Tx2: \( \text{TxInp}[0] \)

- **Value**: 0.05000000 BTC
- **Pkscript**
  - OP_DUP
  - OP_HASH160
  - 45b21c8a0cb687d563342b6c729d31dab58e3a4e
  - OP_EQUALVERIFY
  - OP_CHECKSIG
- **Sigscrip**
  - 304402205846cace0d73de82dfbdeba4d65b9856d7c1b1730eb401cf4906b2401a69b
dc90220589d36d36be64e774c8796b96c011f29768191abeb7f56ba20ff0b0351280860c01
  - 03557c228b080703d52d72ead1bd93fc72f45c4dd4c2b7a20c458e2d069c8dd9e

*from UTXO (Bitcoin script)*

*from TxInp[0]*
A stack machine. Not Turing Complete: no loops.

Quick survey of op codes:
1. **OP_TRUE** (OP_1), **OP_2**, ..., **OP_16**: push value onto stack
   
   81  82  96

2. **OP_DUP**: push top of stack onto stack
   
   118
3. control:

99   **OP_IF** <statements> **OP_ELSE** <statements> **OP_ENDIF**

105 **OP_VERIFY:** abort fail if top = false

106 **OP_RETURN:** abort and fail

what is this for? ScriptPK = [OP_RETURN, <data>]

136 **OP_EQVERIFY:** pop, pop, abort fail if not equal
4. arithmetic:

   OP_ADD, OP_SUB, OP_AND, ...: pop two items, add, push

5. crypto:

   OP_SHA256: pop, hash, push

   OP_CHECKSIG: pop sig, pop pk, verify sig. on Tx, push 0 or 1

6. Time: OP_CheckLockTimeVerify (CLTV):
   fail if value at the top of stack > Tx locktime value.
   usage: UTXO can specify min-time when it can be spent
Example: a common script

\[
\text{stack: } \text{empty} \\
<\text{sig}> <\text{pk}> \\
<\text{sig}> <\text{pk}> <\text{pk}> \\
<\text{sig}> <\text{pk}> <\text{hash}> \\
<\text{sig}> <\text{pk}> <\text{hash}> <\text{pkhash}> \\
<\text{sig}> <\text{pk}> \\
1 \\
\Rightarrow \text{successful termination}
\]

\[
\text{init} \\
\text{push values} \\
\text{DUP} \\
\text{HASH256} \\
\text{push value} \\
\text{EQVERIFY} \\
\text{CHECKSIG} \\
\text{verify}(pk, \text{Tx, sig})
\]
Alice wants to pay Bob 5 BTC:

- **step 1:** Bob generates sig key pair $(pk_B, sk_B) \leftarrow \text{Gen}()$
- **step 2:** Bob computes his Bitcoin address as $addr_B \leftarrow H(pk_B)$
- **step 3:** Bob sends $addr_B$ to Alice
- **step 4:** Alice posts Tx:

```
input 7 BTC

<table>
<thead>
<tr>
<th>5</th>
<th>ScriptPK_B</th>
<th>2</th>
<th>ScriptPK_A</th>
<th>0</th>
</tr>
</thead>
</table>
```

ScriptPK_B:

```
DUP HASH256 <addr_B> EQVERIFY CHECKSIG
```

Point to Alice’s UTXO
Transaction types: (1) P2PKH

“input” contains ScriptSig that authorizes spending Alice’s UTXO

- example: ScriptSig contains Alice’s signature on Tx
  \[\Rightarrow\] miners cannot change ScriptPK_B (will invalidate Alice’s signature)

ScriptPK_B:

\[
\text{DUP HASH256 } \langle \text{addr}_B \rangle \text{ EQVERIFY CHECKSIG}
\]
Transaction types: (1) P2PKH

Later, when Bob wants to spend his UTXO:

create a $T_{x_{\text{spend}}}$

$T_{x_{\text{spend}}}$:

\[
\begin{array}{ccc}
\text{TxID} & 0 & \text{ScriptSig}_B \\
\end{array}
\]

points to UTXO$_B$

\[
\begin{array}{c}
<\text{sig}> \ <\text{pk}_B> \\
\end{array}
\]

(authorizes spending UTXO$_B$)

\[
<\text{sig}> = \text{Sign}(sk_B, \ Tx) \quad \text{where} \quad Tx = (T_{x_{\text{spend}}} \text{excluding all ScriptSigs}) \quad \text{(SIGHASH\_ALL)}
\]

Miners validate that $\text{ScriptSig}_B \mid \text{ScriptPK}_B$ returns true
P2PKH: comments

• Alice specifies recipient’s pk in \( \text{UTXO}_B \)

• Recipient’s pk is not revealed until \( \text{UTXO} \) is spent
  (some security against attacks on pk)

• Miner cannot change \(<\text{Addr}_B>\) and steal funds:
  invalidates Alice’s signature that created \( \text{UTXO}_B \)
ECDSA malleability:
Given \((m, \text{sig})\) anyone can create \((m, \text{sig’})\) with \(\text{sig} \neq \text{sig’}\)

\[ \Rightarrow \] miner can change sig in Tx and change TxID = SHA256(Tx)

\[ \Rightarrow \] Tx issuer cannot tell what TxID is, until Tx is posted

\[ \Rightarrow \] leads to problems and attacks

**Segregated witness:** signature is moved to witness field in Tx
TxID = Hash(Tx without witnesses)
Transaction types: (2) P2SH: pay to script hash

Let’s payer specify a redeem script (instead of just pkhash)

Usage: payee publishes hash(redeem script) ← Bitcoint addr.
      payer sends funds to that address

\[\text{ScriptPK in UTXO: } \text{HASH160 } <\text{H(redeem script)}> \text{ EQUAL} \]

\[\text{ScriptSig to spend: } <\text{sig}_1> <\text{sig}_2> \ldots <\text{sig}_n> <\text{redeem script}> \]

payer can specify complex conditions for when UTXO can be spent
Miner verifies:

(1) `<ScriptSig>` ScriptPK = true ← payee gave correct script

(2) ScriptSig = true ← script is satisfied
Example P2SH: multisig

**Goal**: spending a UTXO requires $t$-out-of-$n$ signatures

Redeem script for 2-out-of-3: (set by payer)

```
<2> <PK_1> <PK_2> <PK_3> <3> CHECKMULTISIG
```

hash gives P2SH address

ScriptSig to spend: (by payee)  
```
<0> <sig1> <sig3> <redeem script>
```
Next lecture: interesting scripts, wallets, and how to manage crypto assets