Decentralization:

Part 1:
Multi-Party Computation /
Zero Knowledge Proofs
Oblivious Transfer

Part 2:
Nakamoto Consensus

Rohit Khera
What We’ll Cover

1 General Protocols
1.1 Goals, Applications, Definitions
1.2 Security Definitions in the MPC Setting
1.3 Zero Knowledge Proofs, Quadratic Residues and the Simulation Paradigm
1.4 More on Simulation Paradigm and Security Definitions in the MPC Setting

2 General Protocols - Constructions
2.1 1-2 Oblivious Transfer
2.2 Yao’s Protocol
2.3 Goldreich, Micali, Wigderson (GMW)*
2.4 Threshold RSA
2.5 Virtual HSM
2.6 Information Theoretic Security
2.7 Universal Composability
Multi-Party Computation (MPC)

One way to approach MPC is from the standpoint of game theory, which (in the classic sense), concerns itself with models of conflict and co-operation among rational parties. This is a framework that has been successfully applied to modeling a variety of games.

Game theory does not, however, directly address the issue of if (and how) games can be efficiently played.

In part, the origins of MPC* lie in demonstrating that in a complexity theoretic sense, all games with honest majority are playable.

Multi-Party Computation (MPC): Security Goals in the Two Party Setting

Two parties with private inputs $x_1$ and $x_2$ respectively, jointly compute a “functionality” $f(x_1, x_2) = (f_1(x_1, x_2), f_2(x_1, x_2))$ such that the first party receives the result $f_1(x_1, x_2)$ and the second party $f_2(x_1, x_2)$

This statement can be generalized to the multi-party setting where the number of parties is greater than two
MPC Applications:

Preventing Satellite Collisions

MPC Applications:

Preventing Satellite Collisions: The Solution

Two parties with private inputs $x_1$ and $x_2$ respectively, jointly compute a “functionality” $f(x_1, x_2) = (f_1(x_1, x_2), f_2(x_1, x_2))$ such that the first party receives the result $f_1(x_1, x_2)$ and the second party $f_2(x_1, x_2)$

The Russians and the Americans with private input satellite trajectories $x_1$ and $x_2$ respectively, jointly compute a functionality $f(x_1, x_2)$ that represents the likelihood of a collision such that their respective satellite trajectories remain private
MPC Applications:
The Danish Sugar Beet Double Auction

As the price increases, demand decreases whereas supply increases
MPC Applications:
The Danish Sugar Beet Double Auction: The Solution

Replace the “trusted” auctioneer (thereby saving on commissions) with an MPC that accepts the following private inputs from the buyers and sellers:

Each seller privately inputs a bid in the form of an ascending list of numbers: “At each possible price per unit, I will sell x quantity.”
Similarly, each buyer privately inputs a bid in the form of a descending list of numbers.
At each price, the MPC adds bids to find total demand and supply in the market at the given price.
“The market clearing price” is the price where demand and supply are equal or as close to equal as possible.
Parties are then able to buy or sell the quantities they were willing to buy or sell at the market clearing price.
Security Definitions (General Protocols)

Recall the Security Goal:

$m$ parties with private inputs $x_1, ..., x_m$ jointly compute a “functionality” $f(x_1, ..., x_m)$ such that the computation maintains the privacy of each party’s input.
Security Definitions (General Protocols)

Can the real protocol be “emulated” by a protocol where the parties provide their inputs to a trusted third party (TTP) which computes the functions on their behalf and provides each of them with the results? This setting is called the “Ideal Model”

We describe this more in the section on Zero Knowledge Proofs and Simulation
Security Definitions: The Simulation Paradigm

Loosely speaking, we define the protocol to be secure if what is learned by an adversary by attacking the real protocol is also feasibly attainable in the “ideal” setting, by attacking the ideal protocol with a TTP.
The Simulation Paradigm and Zero Knowledge Proofs

Zero Knowledge Proofs - Quadratic Residues

\[ P : Prover(Alice) \]
\[ V : Verifier(Bob) \]
\[ x \in \mathbb{Z}_n^* \text{ is a quadratic residue mod } n \text{ if there is a } w \text{ s.t. } x = w^2 \mod n \]

Determining “quadratic residuosity” of \( x \) thought to be computationally difficult

Protocol: Prove the following

Statement \( x \) is a quadratic residue mod \( n \).

Public input \( x, n \)

Prover’s (Alice) private input. \( w \) such that \( x = w^2 \ (\mod n) \).

\[ P \rightarrow V \text{ Alice chooses random } u \leftarrow R \mathbb{Z}_n^* \text{ and sends } y = u^2 \text{ to Bob.} \]

\[ P \leftarrow V \text{ Bob chooses } b \leftarrow_r \{0, 1\} \]

\[ P \rightarrow V \text{ If } b = 0, \text{ Alice sends } u \text{ to Bob. If } b = 1, \text{ Alice sends } w \cdot u \ (\mod n). \]

Verification. Let \( z \) denote the number sent by Alice. Bob accepts the proof in the case \( b = 0, z^2 = y \ (\mod n) \). In the case \( b = 1, \text{ Bob accepts the proof if } z^2 = xy \ (\mod n). \)
Zero Knowledge Proofs - Quadratic Residues

\( \mathbf{QR} \) is the set of quadratic residues in \( \mathbb{Z}_n^* \).
\( \mathbf{P} \) must show that \( x \in \mathbf{QR} \) to the verifier, but does so without providing the root \( w \) s.t. \( x = w^2 \mod n \).

We say \( \mathbf{P} \) proves \( x \in \mathbf{QR} \) “in zero knowledge”.

The Protocol in the prior slide utilizes the fact that \( \mathbf{QR} \) is a group, i.e. if \( x \) and \( y \) are residues then so is \( xy \).

Properties:
(1) Completeness - if \( x \) is really a residue and \( \mathbf{P} \) and \( \mathbf{V} \) follow the protocol \( \mathbf{V} \) accepts the proof with probability 1.

(2) Soundness - if \( x \) is a non-residue, \( \mathbf{V} \) rejects the proof with probability 1/2.

(3) Zero Knowledge Property - How do we describe this more precisely?
Simulation and the Zero Knowledge Property

Making the zero knowledge property more precise, we say the prover $P$ is zero knowledge if for every cheating verifier $V^*$ there is PPT algorithm $S$ (called a simulator for $V^*$) such that for all valid inputs to the QR protocol the following two random variables are computationally indistinguishable:

1. $\langle P, V^* \rangle (x)$ - the output of $V^*$ after interacting with $P$ on common input $x$
2. $S(x)$ - the output of $S$ on the input $x$

$S$ gets the input $x$, but does not interact with $P$
The intuition behind the ZK property is as follows:
If we can construct the Simulator $S$ such that variables (1) and (2) on the prior slide are indistinguishable, then the “real” protocol QR must not leak any knowledge about $w$ since the output of $V^*$ is indistinguishable from the output of the simulation $S(x)$ (recall $x = w^2 \mod n$).

The only party with knowledge of $w$ is $P$ - the fact that a simulator $S$ that is able to produce the same output as $V^*$ without access to $P$ exists, implies that $V^*$ learns nothing about $w$, and accepts the veracity of the statement $x \in QR$ with soundness bound $1/2$. 
Simulation and Security Definitions in the MPC Setting

In the MPC setting, our goals are somewhat different - we’re concerned (among other things) with privacy of the parties’ inputs. In a vein similar to the ZK setting, to prove that an MPC preserves privacy of its inputs, we compare the effect of adversaries that participate in executions of the real protocol, to the effect of adversaries in an imaginary execution of an “ideal” protocol which enlists a “trusted third party” (TTP). The role of the TTP is to implement the functionality of the MPC, accept the parties’ inputs, and compute the result of the functionality on those inputs (while keeping them private). Eventually, the TTP returns the result to all parties. The security argument is then (roughly) that privacy is preserved if the distribution of outputs of the parties in an execution of the real protocol is computationally indistinguishable from the outputs of the parties in the ideal execution.
\( OT_1^2 \): 1-out-of-2 Oblivious Transfer

For a 1-out-of-2 oblivious transfer (\( OT_1^2 \)), the sender with a pair of input messages \((m_0, m_1)\) interacts with a receiver who "obliviously" inputs a "choice" bit \(\sigma\).

Based on this, the receiver learns \(m_\sigma\) without learning anything about \(m_{1-\sigma}\), while the sender learns nothing about \(\sigma\).
**OT$_1^2$**: 1-out-of-2 Oblivious Transfer (Chou and Orlandi 2015)

<table>
<thead>
<tr>
<th>Sender</th>
<th>Receiver</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input: $(M_0, M_1)$</td>
<td>Input: $c$</td>
</tr>
<tr>
<td>Output: none</td>
<td>Output: $M_c$</td>
</tr>
</tbody>
</table>

Concretely, suppose the Receiver wants to obliviously select message $M_1$:

- She sets $c = 1$ and $B = Ag^b$
- such that the sender sets $k_0 = H(B^a) = H((Ag^b)^a)$
- and $k_1 = H((\frac{B}{A})^a) = H((\frac{Ag^b}{A})^a) = H((g^b)^a)$
- $= H(g^{ba}) = H(g^{ab}) = H((g^a)^b) = H(A^b)$

Since the receiver knows $b$ and received $A$ from the sender, she is able to compute $k_1$, but not so for $k_0$ since this requires knowing $a$.

This means that she can only decrypt $e_1 \leftarrow E_{k_1}(M_1)$ to retrieve $M_1$

In theory, the requirement is that we select finitely generated groups.
So then for efficiency, for example, we can choose additive elliptic curve groups over characteristic 2 fields
Yao’s Protocol (Garbled Circuits)

Convert function to a boolean circuit

Alice’s inputs

Bob’s inputs

Truth table

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Yao’s Protocol (Garbled Circuits)

Let's evaluate one gate securely (then we can generalize to the entire circuit)

Alice picks two random keys for each wire into the gate

One key corresponds to “0”, and the other to “1”

Six keys in total for a gate with 2 input wires
Yao’s Protocol (Garbled Circuits)

Alice then encrypts each row of the truth table by encrypting the out wire key with the corresponding pair of input wire keys

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>input wire $w_1$</th>
<th>input wire $w_2$</th>
<th>output wire $w_3$</th>
<th>garbled computation table</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k_1^0$</td>
<td>$k_2^0$</td>
<td>$k_3^0$</td>
<td>$E_{k_1^0}(E_{k_2^0}(k_3^0))$</td>
</tr>
<tr>
<td>$k_1^0$</td>
<td>$k_2^1$</td>
<td>$k_3^1$</td>
<td>$E_{k_1^0}(E_{k_2^1}(k_3^1))$</td>
</tr>
<tr>
<td>$k_1^1$</td>
<td>$k_2^0$</td>
<td>$k_3^1$</td>
<td>$E_{k_1^1}(E_{k_2^0}(k_3^1))$</td>
</tr>
<tr>
<td>$k_1^1$</td>
<td>$k_2^1$</td>
<td>$k_3^1$</td>
<td>$E_{k_1^1}(E_{k_2^1}(k_3^1))$</td>
</tr>
</tbody>
</table>
Yao’s Protocol (Garbled Circuits)

Alice randomly permutes (“garbles”) the rows in the encrypted truth table and sends it to Bob along with the key corresponding to her input bit.

Alice and Bob now run a 1-out-of-2 oblivious transfer protocol in order for Bob to choose the key corresponding to his input bit for the gate.

Alice’s input into the OT are the two keys that are associated with Bob’s wire.

Bob’s input into the OT is the single bit corresponding to the input for his wire.

<table>
<thead>
<tr>
<th>input wire $w_1$</th>
<th>input wire $w_2$</th>
<th>output wire $w_3$</th>
<th>garbled computation table</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k_1^0$</td>
<td>$k_2^0$</td>
<td>$k_3^0$</td>
<td>$E_{k_1^0}(E_{k_2^0}(k_3^0))$</td>
</tr>
<tr>
<td>$k_1^0$</td>
<td>$k_2^1$</td>
<td>$k_3^1$</td>
<td>$E_{k_1^0}(E_{k_2^1}(k_3^1))$</td>
</tr>
<tr>
<td>$k_1^1$</td>
<td>$k_2^0$</td>
<td>$k_3^1$</td>
<td>$E_{k_1^1}(E_{k_2^0}(k_3^1))$</td>
</tr>
<tr>
<td>$k_1^1$</td>
<td>$k_2^1$</td>
<td>$k_3^1$</td>
<td>$E_{k_1^1}(E_{k_2^1}(k_3^1))$</td>
</tr>
</tbody>
</table>
Yao’s Protocol (Garbled Circuits)

Bob, now possesses two keys, one corresponding to his input into the gate and other corresponding to Alice’s input.

Bob can now decrypt a single row in the truth table in order to retrieve one of the output wire keys.

<table>
<thead>
<tr>
<th>input wire $w_1$</th>
<th>input wire $w_2$</th>
<th>output wire $w_3$</th>
<th>garbled computation table</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k_1^0$</td>
<td>$k_2^0$</td>
<td>$k_3^0$</td>
<td>$E_{k_0}^0(E_{k_3}^0(k_3^0))$</td>
</tr>
<tr>
<td>$k_1^0$</td>
<td>$k_2^0$</td>
<td>$k_3^1$</td>
<td>$E_{k_0}^0(E_{k_3}^1(k_3^1))$</td>
</tr>
<tr>
<td>$k_1^1$</td>
<td>$k_2^0$</td>
<td>$k_3^1$</td>
<td>$E_{k_1}^1(E_{k_3}^0(k_3^0))$</td>
</tr>
<tr>
<td>$k_1^1$</td>
<td>$k_2^1$</td>
<td>$k_3^1$</td>
<td>$E_{k_1}^1(E_{k_3}^1(k_3^1))$</td>
</tr>
</tbody>
</table>

Bob decrypts output wire key $k_3^1$
Yao’s Protocol (Garbled Circuits)

Bob can now go ahead and evaluate the entire circuit. For each intermediate wire, Bob retrieves a key but cannot know (and should not) know whether this key corresponds to a 0 or 1. Bob finally retrieves the key for the final output wire and conveys this to Alice, who tells him whether it corresponds to a 0 or 1.
Yao’s Protocol (Garbled Circuits)

consider the case that the input to the circuit is $a = 1$, $b = 0$, $c = 1$ and $d = 1$.

output

$f = 0$ and $g = 1$

input

1011 or 0001 or 1100 or 1110
Yao’s Protocol (Garbled Circuits)

1) Communication Complexity - Yao is a constant round protocol - one oblivious transfer for each of Bob’s input bits

2) Number of communication rounds is therefore independent of the number of gates or depth of the circuit

3) Recent approaches utilizing functional encryption and the LWE assumption that allow re-use of circuits (“Reusable Garbled Circuits and Succinct Functional Encryption”- Goldwasser, Kalai, Ada Popa, Vaikuntanathan, Zeldovich)
Goldreich, Micali, Wigderson (GMW)*

Supports multiple parties ($n > 2$)

Parties propagate “shares” of bits through NOT and AND gates (jointly evaluating the gates)
Goldreich Micali Wigderson (GMW)

Consider \( n \) parties computing a circuit of NOT gates and AND gates \((n > 2)\)

Use a simple secret-sharing scheme such that a bit \( b \) is shared by a random sequence of \( m \) bits that sum up to \( b \text{ mod } 2 \). Each party shares each of its input bits with all parties (by sending each party a random value and setting its own share accordingly)

Each \( P_i \) shares each bit \( b \) of their input with all the other parties as follows:

1. Choose \( n \) random bits \( b_1, ..., b_n \) such that \( b = \sum_{i=1}^{n} b_i \text{ mod } 2 \).
2. Send share \( b_j \) to \( P_j \) (including one share to one’s self).
A NOT gate has an input $b$ and an output $c = b + 1$. The input $b = \sum b_j$ (the sum of shares $b_i$ held respectively by party $P_i$) and we want one output $c = \sum c_i$, where each share $c_i$ is held by party $P_i$.

(All sums are mod 2)

- $P_1$ sets $c_i = b_i + 1 \mod 2$
- $\forall i = 2, ..., n$, $P_i$ sets $c_i = b_i$

Which party is chosen as $P_1$ is unimportant. It is only necessary that one of the parties be designated as $P_1$. 
AND gate is computed by noting that it is multiplication $\mod 2$ (i.e., $c \text{ AND } d = c \cdot d \mod 2$)

Requires parties to carry out a 1-out-of-4 oblivious transfer

Parties jointly scan the circuit from its input wires to the output wires, processing each gate jointly

At the end of the computation, each party holds a share of each output wire. The desired output is obtained by letting each party send its share of each output wire to all parties
<table>
<thead>
<tr>
<th>Function</th>
<th>No. ANDs</th>
<th>No. XORs</th>
<th>No. INV</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES (No Key Expansion)</td>
<td>6800</td>
<td>25124</td>
<td>1692</td>
</tr>
<tr>
<td>AES (Key Expanded)</td>
<td>5440</td>
<td>20325</td>
<td>1927</td>
</tr>
</tbody>
</table>

https://www.cs.bris.ac.uk/Research/CryptographySecurity/MPC/
OT Extension

OT is provably secure using public-key assumptions (eg. computational Diffie Hellman in Chou & Orlandi 2015)

OT can be extended to use cheaper symmetric primitives (PRFs, hashing) after a fixed number of seed OT’s

OT “analag” to hybrid encryption
Universal Composability

1. A method for constructing protocols

2. Protocol is said to ‘UC realize’ some functionality $f$ - protocol is then said to be “UC secure”

3. UC secure protocols maintain their security properties if they are composed with other UC secure protocols

4. Offers security for concurrent protocol execution since UC secure protocol can be composed with itself
Virtual Hardware Security Module (vHSM)

Recall scenario of two parties jointly computing the likelihood of satellite collision while maintaining the privacy of their input satellite trajectories.

Two parties with private inputs $x_1$ and $x_2$ respectively, jointly compute a "functionality" $f(x_1, x_2) = (f_1(x_1, x_2), f_2(x_1, x_2))$ such that the first party receives the result $f_1(x_1, x_2)$ and the second party $f_2(x_1, x_2)$.

The Russians and the Americans with private input satellite trajectories $x_1$ and $x_2$ respectively, jointly compute a functionality $f(x_1, x_2)$ that represents the likelihood of a collision such that their respective satellite trajectories remain private.
Virtual Hardware Security Module (vHSM)

Consider a two way key splitting scheme $G(K) \rightarrow k_1, k_2$ that generates key shares $k_1, k_2$ from a master key $K$ (Can be generalized to $n$ way splitting)

Two parties with private inputs $x_1$ and $x_2$ respectively, jointly compute a “functionality” $f(x_1, x_2) = (f_1(x_1, x_2), f_2(x_1, x_2))$ such that the first party receives the result $f_1(x_1, x_2)$ and the second party $f_2(x_1, x_2)$

Two (or more) servers with private key splits $k_1$ and $k_2$, jointly compute an encryption (or decryption) function $\text{Enc}(k_1, k_2, \cdot)$ such that their key shares remain private
Threshold RSA (A Simple MPC for RSA)

The RSA public key is a pair \((e, N)\) and the private key is a pair \((d, N)\) with \(N = p \cdot q\) with \(p\) and \(q\) being randomly generated prime numbers. The parameters \(e\) and \(d\) are chosen such that \(e \cdot d = 1 \mod (p - 1)(q - 1)\).

RSA decryption of a ciphertext \(y\) is \(y^d \mod N\) (“Raw” RSA)
Threshold RSA (A Simple MPC for RSA)

Choose \( d_1 \) and \( d_2 \) randomly such that \( d_1 + d_2 = d \).

Alice and Bob would then hold the private key shares \((d_1, N), (d_2, N)\) respectively.

To decrypt a ciphertext \( y \), Alice and Bob would independently compute \( x_1 = y^{d_1} \mod N \) and \( x_2 = y^{d_2} \mod N \) and send the results to one another.

Then the product \( x_1 \cdot x_2 = x \) is the decryption of \( y \) since \( x_1 \cdot x_2 = y^{d_1 + d_2} \mod N = y^d \mod N \).

Thus Bob and Alice have decrypted the message without revealing their shares of the key and the full decryption key “never appears in the clear anywhere.”
Threshold RSA (A Simple MPC for RSA)  
Re-randomizing Shares

Alice and Bob run a commitment protocol to agree on a random number $r$.

Alice sets her share equal to $d'_1 = d_1 + r$ and Bob sets his share equal to $d'_1 = d_1 - r$

All of the math still works since $d'_1 + d'_2 = d$

This allows Alice and Bob to rotate their shares
1. Damgard, Pastro, Smart and Zakarias ("Multiparty Computation from Somewhat Homomorphic Encryption")

2. Supports dishonest majorities with active corruption of \(n - 1\) computation nodes (\(n > 2\))

3. Homomorphic encryption scheme utilized is BGV (Brakerski, Gentry, Vaikuntanathan)

4. BGV is based on Ring-LWE assumption instantiated on integer subrings \(\mathcal{O}_{Q(\zeta_m)} = \mathbb{A}_p = \mathbb{Z}[X]/(\Phi_m(X), p) \cong \mathbb{Z}_p(\zeta_m)\) of cyclotomic fields \(Q(\zeta_m)/Q\)

5. Don’t need all of BGV machinery (field automorphisms etc) in the context of SPDZ since only need to address BGV circuits of multiplicative depth one
Nakamoto Consensus
The Damgard Merkle Transform

For a hash function $H(\cdot)$, we define a collision as two messages $m_1$ and $m_2$ such that $H(m_1) = H(m_2)$

$H(\cdot)$ is collision resistant, if it is hard to find two messages that collide

![Diagram](image)

let $f$ be a fixed length compression function that maps inputs of length $2n$ to outputs of length $n$

Construct an iterated hash function $H(\cdot)$ (that handles messages of arbitrary length) from the compression function $f$ as depicted above

**Theorem** If $f(\cdot)$ is collision resistant, then so is $H(\cdot)$
Hash Pointer

A pointer to some data along with a hash of the data
An append only hash linked list with hash pointers. The top hash is stored in some “immutable” fashion, i.e. it can be digitally signed or published in a daily newspaper.
Block Chain (Security)

Block chain construction is similar to that of the Damgard Merkle transform.

Security against block forgery boils down to fact that it is hard to find block hash collisions so long as the underlying hash function (used to construct the hash pointers) is collision resistant.
Merkle Tree

An efficient means for determining “Proof of Membership” for blocks
Identity

Identity in bitcoin is the hash of an ECDSA public key (this is also called an “address”)

ECDSA is defined over curve secp256r1 (?)
Note current NSA Guidance is to use the p384 curve

(https://ringcipher.com/2016/04/14/prime-curves-post-snowden/)
A Simple Crypto Currency

The Authority can mint a coin at will through the Create-a-Coin() method

This protocol does not protect against double spending
Simple Crypto Currency Revisited

To solve the double spending problem, the authority establishes an append only block chain ledger containing valid transactions.

A transaction is only accepted once it has been recorded in the ledger

The authority validates all the transactions by signing the top hash pointer
Payment Transaction
Replacing the Central Authority with a Consensus Protocol

In the previous example, the central authority validated all transactions in order to address the double spend problem. We can replace the authority with a consensus protocol that has the following features:

1. The payer broadcasts a payment to the network
2. “Miner” nodes listen for payment transactions for inclusion into a block
3. The miner collects a prescribed number of transactions to make a block
4. Miner submits a block for other nodes to accept
5. Nodes accept the new block only if it contains legitimate transactions (i.e., the payed coins have not already been spent and the signatures are from valid “addresses”)
6. An accepted block’s hash is then included in the next block that is created
Double Spend Attempt

Alice Pays Bob through the transaction $C_A \rightarrow B$ but then attempts to create an earlier transaction $C_A \rightarrow A'$ where $A'$ is a sybil identity controlled by Alice in an attempt to invalidate the transaction $C_A \rightarrow B$
Mining and Proof of Work

In order to create a block, a node or “miner” needs to spend cycles to compute a nonce such that the following hash

\[ H(\text{nonce}||\text{prior\_block\_hash}||\text{transaction}_1||\text{transaction}_2||...) \]

is less than some target number. Given collision / pre-image resistance and pseudo random nature of the hash function output, this is not a trivial computation and requires brute force search for the nonce value. This is called a “hash puzzle”
Double Spend Attempt

The protocol says that for events that lead to forks in the chain, miners should always append new blocks to the longest fork (thereby legitimizing this fork)
Double Spend Attempt

In order for the double spend to work, Alice must create a longer “fraudulent” chain than the current chain. But this is computationally expensive making it hard for her fraudulent chain to catch up / surpass the legitimate chain in length.
Double Spend Attempt

From Bob’s viewpoint, one option is that he should only except payment (thereby providing goods or services to Alice) once the transaction $C_A \rightarrow B$ has been “confirmed” by its inclusion in the longest chain and is a few blocks deep into this chain.
Double Spend Attempt

Double spend probability decreases exponentially with each successive block confirmation of the legitimate transaction.

A common practice is to allow for 6 confirmations in order ensure transaction winds up on the long term consensus chain.
Question (from a web forum)
We want spenders to have certainty that their transaction is valid at the time it takes a spend to flood the network, not at the time it takes for branch races to be resolved.

Satoshi’s Response
Instant non-repudiability is not a feature, but it’s still much faster than existing systems. Paper cheques can bounce up to a week or two later. Credit card transactions can be contested up to 60 to 180 days later. Bitcoin transactions can be sufficiently irreversible in an hour or two.
Mining

Block Reward

Transaction fees
# Mining - Block Reward

## Bitcoin Block Reward Halving Countdown

![Halving Countdown](image)

Reward-Drop ETA date: 10 Jul 2016 23:16:50

The Bitcoin block mining reward halves every 210,000 blocks, the coin reward will decrease from 25 to 12.5 coins.

<table>
<thead>
<tr>
<th></th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total Bitcoins in circulation:</td>
<td>15,622,900</td>
</tr>
<tr>
<td>Total Bitcoins to ever be produced:</td>
<td>21,000,000</td>
</tr>
<tr>
<td>Percentage of total Bitcoins mined:</td>
<td>74.39%</td>
</tr>
<tr>
<td>Total Bitcoins left to mine:</td>
<td>5,377,100</td>
</tr>
<tr>
<td>Total Bitcoins left to mine until next blockhalf:</td>
<td>127,100</td>
</tr>
<tr>
<td>Bitcoin price (USD):</td>
<td>$575.72</td>
</tr>
<tr>
<td>Market capitization (USD):</td>
<td>$8,994,415,968.00</td>
</tr>
<tr>
<td>Bitcoins generated per day:</td>
<td>3,600</td>
</tr>
</tbody>
</table>

Credit: [http://www.bitcoinblockhalf.com/](http://www.bitcoinblockhalf.com/)
Hash target adjusted so that the network on average mines a block every 10 minutes

For a miner:
average time to next block = \frac{10 \text{ minutes}}{\text{miner's fraction of network hash power}}
// Internal miner

// Scannash scans nonces looking for a hash with at least some zero bits.
// The nonce is usually preserved between calls, but periodically or if the
// nonce is 0xffffffff or above, the block is rebuilt and nonce starts over at
// zero.

bool static ScanHashconst (BlockHeader *pblock, uint32_t nNonce, uint128_t *phash)
{
    // Write the first 76 bytes of the block header to a double-SHA256 state.
    CHash256 hasher;
    CDataStream ss(SER_NETWORK, PROTOCOL_VERSION);
    ss << *pblock;
    assert(ss.size() == 80);
    hasher.Write((unsigned char*)&ss[0], 76);

    while (true)
    {
        nNonce++;

        // Write the last 4 bytes of the block header (the nonce) to a copy of
        // the double-SHA256 state, and compute the result.
        CHash256(hasser).Write((unsigned char*)&nNonce, 4).Finalize((unsigned char*)phash);

        // Return the nonce if the hash has at least some zero bits,
        // caller will check if it has enough to reach the target
        if (((uint64_t*)phash)[15] == 0)
            return true;

        // If nothing found after trying for a while, return -1
        if ((nNonce & 0xffffffff) == 0)
            return false;
    }
}
Non Cooperative Games

An open question:

Does the protocol represent a Nash Equilibrium for bitcoin participants?
Early GPU Based Mining

OpenCL implementation

Fully unrolled SHA256 (64 rounds)

Every thread attempts to compute hash value within target range based on a unique nonce derived from thread id

Successful nonces are flagged for driver code
Memory Hard Puzzles

SHA256 not memory hard since 256 bit state easily accommodated in registers (Intel SSE etc.)

To discourage hardware based mining there is a trend to switch to memory intensive hashing such as scrypt

Ethereum (discussed later) uses the Keccak function with 1600 bit state
Mining Difficulty

Credit: Bitcoin and The Age of Bespoke Silicon, Michael Bedford Taylor University of California, San Diego
16nm ASIC Chip

Unmatched 0.1J/GH efficiency

Amazing 100GH/s* speed

Welcome to the future of mining.

BitFury custom designed new 16nm SHA256 ASIC to deliver the maximum performance and efficiency possible with 16nm technology. Boasting 8162 hash cores on a single die – it is the best solution as the block reward halving approaches the summer of 2016.
Ethereum is a blockchain based decentralized virtual machine that executes programs known as “smart” contracts.

Ethereum can be described formally through the following recurrence:

$$\sigma_{t+1} = \Upsilon(\sigma_t, T)$$

Where:

$\sigma$ is the “world” state, a mapping of Ethereum addresses to accounts and their storage.

$\Upsilon$ is the Ethereum state transition function.
Ethereum

In bitcoin, the world state is a mapping between accounts and addresses where the state of an account is just the balance of the account in BTC.
Also, in bitcoin the state transition function essentially consists of two functions (i) payment (ii) bitcoin creation.

In the ethereum model, the state transition function $\Upsilon$ allows for arbitrary computation and $\sigma$ allows components to store arbitrary state between computations.
Ethereum Smart Contracts

As mentioned, with bitcoin, a transaction is typically a payment or transfer of coins from one address to another, in ethereum, a transaction can have code associated with it (the smart contract) that must be executed as a part of the transaction itself.

From the Ethereum “Yellow Paper”:

A commonly asked question is "where" contract code is executed, in terms of physical hardware. This has a simple answer: the process of executing contract code is part of the definition of the state transition function, which is part of the block validation algorithm, so if a transaction is added into block B the code execution spawned by that transaction will be executed by all nodes, now and in the future, that download and validate block B.
Ethereum

Smart contract programming languages:

Serpent

LLL

Solidity
A commonly asked question is "where" contract code is executed, in terms of physical hardware. This has a simple answer: the process of executing contract code is part of the definition of the state transition function, which is part of the block validation algorithm, so if a transaction is added into block B the code execution spawned by that transaction will be executed by all nodes, now and in the future, that download and validate block B.
Discussion

Founding Digital Currency on Secure Computation (MPC)

(Karim El Defrawy, Joshua Lampkins)