Modern Cryptography Concepts: Hype or Hope

Radia Perlman
DellEMC
Radia.Perlman@dell.com
With only 45 minutes…

• I’m going to go quickly
With only 45 minutes…

• I’m going to go quickly
• Skip details of the math, and certainly no security proofs
With only 45 minutes...

• I’m going to go quickly
• Skip details of the math, and certainly no security proofs
• What problem is being solved, intuition behind how to solve it, whether it’s practical
With only 45 minutes…

- I’m going to go quickly
- Skip details of the math, and certainly no security proofs
- What problem is being solved, intuition behind how to solve it, whether it’s practical
- I gave a much longer version to colleagues
  - With many more topics
  - I had them vote on the topics they found most interesting…so blame them for the selection of topics
Topics

- I’m going to cover the ones in red in the next slide
- The ones in black were less popular with my test audience
Topics

- How to share a secret
- Zero Knowledge Proofs
- Non-interactive ZKP
- Blind signatures, blind decryption
- Oblivious transfer
- Group signatures, ring signatures
- Bit commitment, coin flip
- Making an unfair coin fair
- Circuit model
- Random oracle model
- Identity based encryption
- Secure multiparty computation (VERY brief) just what problem is being solved
- Homomorphic encryption (again, VERY brief)
- Bitcoin/blockchain (not so much crypto, but there is SO much hype)
Sharing a Secret
Sharing a Secret

• Problem: You want to make n backups of a secret S
  – You want to retrieve S in case you forget it
  – But you’re afraid some of the backups might get broken into, and you don’t want S stolen
Sharing a Secret

• Problem: You want to make n backups of a secret S
  – You want to retrieve S in case you forget it
  – But you’re afraid some of the backups might get broken into, and you don’t want S stolen

• Break a secret S into n “shares”, such that retrieval of at least k of the shares reveals S, and retrieval of fewer than k reveals nothing
Sharing a Secret S

• N locations, quorum = 1
  – Trivial…just store S in each location
Sharing a Secret

• n locations, quorum = n
  – Also easy
  – Store n-1 random #’s in n-1 places, and XOR of S and all the random numbers in the nth place

\[ X = S \oplus R_1 \oplus R_2, \ldots \oplus R_7 \]
What about $1 < k < n$?
Secret Sharing with k=2

• Choose a line. Any two points reveal the line, any one point gives no information
• Let S be where it crosses the Y axis
Secret Sharing with \( k=2 \)

- Do this as follows:
  - Choose random number \( b \)
  - Line equation is \( y = bx + S \)
  - \( S = \) value of the line at \( x=0 \)
Secret Sharing of $S$ with $k=2$

Shares are $(i, y_i)$, $(j, y_j)$, $(v, y_v)$, $(l, y_l)$, and $(m, y_m)$.
Knowing any two of those, you can calculate the line, and solve for $S$. 

$y_i$ when $x=i$

$y_j$ when $x=j$

$y_l$ when $x=l$

$y_v$ when $x=v$

$y_m$ when $x=m$

$y = S$ when $x=0$
Secret Sharing of \( S \) with \( k=2 \)

\[ y=S \text{ when } x=0 \]

\[ y_i \text{ when } x=i \]

\[ y_j \text{ when } x=j \]

\[ y_k \text{ when } x=k \]

\[ y_v \text{ when } x=v \]

\[ y_l \text{ when } x=l \]

\[ y_m \text{ when } x=m \]

\[ y_n \text{ when } x=n \]

\[ y_p \text{ when } x=p \]

\[ y_q \text{ when } x=q \]
Secret Sharing with $2 < k < n$

- Create an equation of order $k-1$
- Each share is the value of $y$ at some point
- Given any $k$ points, the equation can be solved and $S$ found
- The math:
  - Choose $k-1$ random numbers $b_1, b_2, \ldots b_{k-1}$
  - Graph is $y = S + b_1x + b_2x^2 + \ldots b_{k-2}x^{k-2} + b_{k-1}x^{k-1}$
  - Secret $S$ is value of $y$ at $x=0$
Notes on secret sharing

• Real numbers are annoying
  – Even integers are kind of annoying (since they have unbounded size)
  – Solution: do arithmetic mod some prime $p > S$

• A share is an $(x, y)$ pair, where $x$ and $y$ are integers mod $p$
  – But you can have fixed $x$-values for each participant
    • Participant 1 is given the value of $y$ at 1
    • Participant 2 is given the value of $y$ at 2
    • Etc.
Summary of secret sharing

• Not only mathematically cool, but very useful and practical
Bit Commitment
Bit Commitment

• Problem to be solved
Bit Commitment

• Problem to be solved
  – Alice and Bob are getting a divorce
Bit Commitment

• Problem to be solved
  – Alice and Bob are getting a divorce
  – They can only talk over the phone
Bit Commitment

• Problem to be solved
  – Alice and Bob are getting a divorce
  – They can only talk over the phone (restraining orders?)
Bit Commitment

• Problem to be solved
  – Alice and Bob are getting a divorce
  – They can only talk over the phone (restraining orders?)
  – They have to decide who gets to keep the house
Bit Commitment

• Problem to be solved
  – Alice and Bob are getting a divorce
  – They can only talk over the phone (restraining orders?)
  – They have to decide who gets to keep the house
  – They will flip a coin
How to Flip a Coin
How to Flip a Coin

• Two obvious protocols
  – First obvious protocol
    • Alice flips the coin
    • Bob calls “heads” or “tails”
    • Alice declares who won the house
How to Flip a Coin

• Two obvious protocols
  – First obvious protocol
    • Alice flips the coin
    • Bob calls “heads” or “tails”
    • Alice declares who won the house
  – Second obvious protocol
    • Alice flips the coin
    • Bob decides which he wants (“heads” or “tails”, but does not reveal his choice to Alice)
    • Alice tells Bob whether it was heads or tails
    • Bob declares whether he won or lost

• Obviously neither of these works (Alice can cheat in the first one, Bob can cheat in the 2nd one)
Review hashes

• Properties of cryptographic hash $h$
  – Infeasible to find two numbers $x, y$ that hash to the same value
    • $(x, y, \text{ such that } h(x) = h(y))$
  – Infeasible, given a hash, to find a number that hashes to that value
    • $(\text{given } H, \text{ find } x \text{ with } h(x) = H)$
The Solution

**Bob**

Choose random # R  
with bottom bit=0 for Tails  
with bottom bit=1 for Heads  
Sends h(R)

**Alice**

Flips coin

---

h(R)

Reveal whether flip was “heads” or “tails”

R
Circuit Model
Why talk about circuit model?

• It’s necessary for understanding secure multiparty computation, and homomorphic encryption
Circuit Model

- Any function of fixed size input and output can be built with a circuit with just two operations
  - XOR (⊕) (addition)
  - AND (&) (multiplication)
Circuit Model

• Any function of fixed size input and output can be built with a circuit with just two operations
  – XOR (⊕) (addition)
  – AND (&) (multiplication)
• This will be useful for proving it’s possible to do homomorphic encryption or secure multiparty computation
• Both of these involve doing computation on data
• If you turn your program into a circuit, then all you have to do is show how to do XOR and AND
Programs as circuits

• You have to expand the circuit for each branch
• Can’t do infinite loops
• Imagine a program that searches a database to retrieve an item with a particular value. With a “real program” you could do binary search, but with a circuit:
  – You have to know the maximum size of the database
  – Your circuit-program has to be written out, and each potential branch must be executed for each item in the maximum sized database
But think about how inefficient this is!

- About a million times slower to do a circuit than a “normal” program,
- Plus, whenever you might have done a branch, you have to execute all possible branches
  - Anything that chooses from a large set must execute on every item
Secure Multiparty Computation
Secure Multiparty Computation

• Problem
  – $n$ participants each have an input
  – Desire to compute a function of all the inputs
  – Where each participant’s input stays secret
Secure Multiparty Computation

• Problem
  – $n$ participants each have an input
  – Desire to compute a function of all the inputs
  – Where each participant’s input stays secret

• Example: An auction
  – Who has the highest bid?
  – Nobody wants their bid revealed to others
Obvious Solution

• A trusted 3\textsuperscript{rd} party who receives all the inputs, computes the answer, and then tells everyone the answer
  – “the highest bid was X”
  – “Joe, you are committed to paying X”
Obvious Solution

• A trusted 3rd party who receives all the inputs, computes the answer, and then tells everyone the answer

• However, cryptographers want to not depend on a trusted 3rd party
Obvious Solution

• A trusted 3rd party who receives all the inputs, computes the answer, and then tells everyone the answer
• However, cryptographers want to not depend on a trusted 3rd party
• Several proposed solutions (all very expensive!)
Obvious Solution

• A trusted 3\textsuperscript{rd} party who receives all the inputs, computes the answer, and then tells everyone the answer
• However, cryptographers want to not depend on a trusted 3\textsuperscript{rd} party
• Several proposed solutions (all very expensive!)
• One is actually deployed in the yearly Danish sugar beet auction
Outline of one solution

• Each participant $P_j$ does secret-sharing of its own input $I_j$, into $n$ $k$-shares (quorum $k+1$)
  – A $k$-share is a point on the graph of a degree $k$ polynomial
  – Shares of participant $J$’s input, $I_j$ are $I_{j1}$, $I_{j2}$, $I_{j3}$, … $I_{jn}$

• Participant $j$ gives
  – $I_{j1}$ to participant $P_1$
  – $I_{j2}$ to participant $P_2$
  – etc
Each participant will now know

- n things: a k-share of each of the n inputs
- Participant 3, for instance, will know
  - $I_{13}, I_{23}, I_{33}, I_{43}, \ldots I_{n3}$
Note

• If \( k+1 \) participants conspired, they could calculate all the inputs

• So the assumption is that this won’t happen
Outline of one solution

- Turn the program into a circuit with operations
  - addition
  - multiplication
- Each participant operates on their own share of the inputs
- At the end of the computation, each participant will have a $k$-share of the answer
Adding two inputs

- If the program says to compute $I_5 + I_7$, each participant adds its share of $I_5$ to its share of $I_7$
  - For instance, participant $m$ computes $I_{5m} + I_{7m}$
  - Result $(I_{5m} + I_{7m})$ is participant $m$’s $k$-share of $I_5 + I_7$
  - Because what’s being computed is shares of the sum of the polynomials for $I_5$ and $I_7$
  - That’s really cool!!!
What about multiplying two inputs?

• Suppose instead the program says to multiply: $I_5 \times I_7$
• Just multiply your share of $I_5$ to your share of $I_7$
• Now everyone has a share of $I_5 \times I_7$ but unfortunately it’s a 2k share (a degree 2k polynomial)!
  – Because the polynomial for $I_5$ is being multiplied by the polynomial for $I_7$
  – Meaning quorum is now 2k+1
• And you’re dead if the polynomial degree is ever bigger than n
Suppose there were a trusted 3rd party Z

- Then everyone could send Z their 2k-shares of $I_5 * I_7$
- Z could calculate $I_5 * I_7$
- And calculate n k-shares of $I_5 * I_7$
- And send one k-share to each member
Suppose there were a trusted 3\textsuperscript{rd} party Z

- Then everyone could send Z their 2k-shares of $I_5 \times I_7$
- Z could calculate $I_5 \times I_7$
- And calculate n k-shares of $I_5 \times I_7$
- And send one k-share to each member
- But if we have a trusted 3\textsuperscript{rd} party we don’t need any of this complicated stuff!!
However....

- The computation the trusted 3\textsuperscript{rd} party needs to do to create k-shares out of the 2k-shares only involves additions ("linear operations")
- And those can be done in a distributed way, on k-shares of the 2k-shares
However….

- The computation the trusted 3\textsuperscript{rd} party needs to do to create k-shares out of the 2k-shares only involves additions ("linear operations")
- And those can be done in a distributed way, on k-shares of the 2k-shares
- Very expensively, of course
  - Both in computation and network bandwidth
  - Because after each multiply, everyone has to distribute k-shares of their 2-k share, so everyone can participate in reducing it, and then distributing k-shares
Summary of Secure Multiparty Computation

• (my opinion)
  – Not a very important problem
  – And sufficiently expensive it’s impractical (despite Danish sugar beet auction)

• But lots of published papers
Homomorphic Encryption
Homomorphic Encryption

• Encrypt your data
• Do operations on the encrypted data
• The answer is the encrypted answer
Homomorphic Encryption

• Sounds really appealing
  – Store encrypted data in a public cloud you don’t really trust
  – Efficiency: Instead of downloading all your data and decrypting it locally (more trusted environment), do operations in the cloud without the cloud knowing the data decryption key
Fully homomorphic (FHE)

- Means you can do any computation on encrypted data without knowing the key
  - And the result will be the encrypted answer
Fully homomorphic (FHE)

• Means you can do any computation on encrypted data without knowing the key
  – And the result will be the encrypted answer
• Don’t get too excited: It’s completely impractical
  – Encrypted data and computation are about 5 or 6 (decimal) orders of magnitude less efficient than doing computation on plaintext
  – Plus of course converting the algorithm to a circuit makes things even worse
What about “partially homomorphic”

• Sure…examples
  – Equality Preserving
    • Good security practice uses an “IV” for each encrypted item, so that the same plaintext will encrypt differently each time
What about “partially homomorphic”

- Sure…examples
  - Equality Preserving
    - Good security practice uses an “IV” for each encrypted item, so that the same plaintext will encrypt differently each time
    - But if you want to test for equality, skip the IV
      - Then you can test for equality (the same plaintext will always have the same ciphertext)
What about “partially homomorphic”

• Sure…examples
  – Equality Preserving
    • Good security practice uses an “IV” for each encrypted item, so that the same plaintext will encrypt differently each time
    • But if you want to test for equality, skip the IV
      – Then you can test for equality (the same plaintext will always have the same ciphertext)
  – Order-preserving Encryption
    • A cute idea (next slide)

• These leak information obviously
Order-Preserving Encryption

• To encrypt $n$ items, sorted from smallest to largest:
  – Encrypt each one (say using AES), to produce a 128-bit result
  – Create, say, a 256-bit result by putting the (sorted) item number in the high order part of the result
# Order-Preserving Encryption

<table>
<thead>
<tr>
<th>plaintext</th>
<th>Order-preserving Encryption</th>
</tr>
</thead>
<tbody>
<tr>
<td>79</td>
<td>0000001x^&amp;*bd</td>
</tr>
<tr>
<td>215</td>
<td>0000002qj&amp;*kl</td>
</tr>
<tr>
<td>218</td>
<td>00000034j^%$c</td>
</tr>
<tr>
<td>979</td>
<td>0000004n$FOjw</td>
</tr>
<tr>
<td>7933</td>
<td>0000005!Jilr7r</td>
</tr>
</tbody>
</table>
Order-Preserving Encryption

• Can get fancier if you need to be able to add new things
  – Start with first item, put a medium number for high order part
  – To insert an item, choose a high-order part midway between already-inserted items
  – Worst case: if the items are already sorted
Order-Preserving Encryption
(where you can add items later)

<table>
<thead>
<tr>
<th>plaintext</th>
<th>Order-preserving Encryption</th>
</tr>
</thead>
<tbody>
<tr>
<td>79</td>
<td>000373x^&amp;*bd</td>
</tr>
<tr>
<td>215</td>
<td>005762qj&amp;*kl</td>
</tr>
<tr>
<td>218</td>
<td>0066914j^%$c</td>
</tr>
<tr>
<td>979</td>
<td>073118n$FOjw</td>
</tr>
<tr>
<td>7933</td>
<td>091326!Jilr7r</td>
</tr>
</tbody>
</table>
Partially Homomorphic, minimizing information leakage

• Let’s say you want only authorized people to be able to test for equality

• Do equality-preserving encryption, but then encrypt one more time, with a key available only to people authorized to test for equality
# Order-Preserving Encryption

<table>
<thead>
<tr>
<th>Plaintext</th>
<th>Order-preserving Encryption</th>
<th>Encrypt with key $K$, known if authorized to see ordering</th>
</tr>
</thead>
<tbody>
<tr>
<td>79</td>
<td>000001x^&amp;*bd</td>
<td>${000001x^&amp;<em>bd}K = 49^</em>^&amp;^{^%ac}$</td>
</tr>
<tr>
<td>215</td>
<td>000002qj&amp;*kl</td>
<td>${000002qj&amp;<em>kl}K = S7^</em>^#xje$</td>
</tr>
<tr>
<td>218</td>
<td>0000034j^%$c</td>
<td>${0000034j^%$c}K = &amp;8:lp#xu$</td>
</tr>
<tr>
<td>979</td>
<td>000004n$FOjw</td>
<td>${000004n$FOjw}K = z]9^#x87$</td>
</tr>
<tr>
<td>7933</td>
<td>000005!Jilr7r</td>
<td>${000005!Jilr7r}K = Ju7^*7v33$</td>
</tr>
</tbody>
</table>
Back to Fully Homomorphic Encryption
Turn your program into a circuit

• Remember, any computation can be built with a circuit with just two operations
  – XOR (⊕) (multiplication)
  – AND (&) (addition)

• Then figure out a homomorphic scheme that can do both operations
One homomorphic operation

- Unpadded RSA, multiplication
- Public key is (e,n), private key is (d,n)
- Encryption of x is $x^e \mod n$
- Want $x \times y$
- Multiply encrypted versions:
  - $(x^e \mod n) \times (y^e \mod n) = (xy)^e \mod n$
- Answer is encrypted $x \times y$
**Fully Homomorphic**

- All the known schemes involve “noisy numbers”
- Intuition
  - Encrypted number is “near” a legal value
  - If the encrypted number was *exactly* a legal value, the scheme wouldn’t be secure
  - Adding or multiplying numbers increases the noise
  - If the noise gets too much, it can’t decrypt properly

- So there were schemes that allowed doing a few operations on the encrypted data, but then the noise got too great
Breakthrough: “Bootstrapping”

• Craig Gentry made this breakthrough
• It is possible to “refresh” the encrypted data to the original noise level
  – Encrypt, then homomorphically decrypt the data
• Really really slow
Summary of Fully Homomorphic Encryption

• Lots of papers
• Absolutely impractical in terms of computation and data expansion
Bitcoin/Blockchain
What is Blockchain?
What is Blockchain?

• It’s a word
What is Blockchain?

• It’s a word
  – “When I use a word, it means just what I choose it to mean”
    Humpty Dumpty, in Alice in Wonderland’s Through the Looking Glass
What is Blockchain?

• It’s a word
  – “When I use a word, it means just what I choose it to mean”
    Humpty Dumpty, in Alice in Wonderland’s Through the Looking Glass

• Extreme confusion because people are using the term “blockchain” for lots of things
Hype

• Articles about how “it” (whatever “it” is) is being considered for all sorts of problems
  – IoT
  – Alternative to PKI for managing identities
  – Real estate transactions
  – Protecting our nuclear technology (really!)
    • “Even the US Military is looking at blockchain technology to secure nuclear weapons"
Bitcoin’s Blockchain

• The design of the blockchain technology as invented for Bitcoin requires (a lot of) monetary compensation to a community of “miners”
Bitcoin
Bitcoin

- Paper in 2008, by Satoshi Nakamoto (presumably a pseudonym)
- Released as open source software soon after
- The community agrees on modifications, so details of Bitcoin might change
- Really no spec…just the code.
- As we’ll see, there could be (and has been) a problem if different implementations deployed, with subtle incompatibilities
Bitcoin Design Goals

• Don’t trust known institutions
• Instead, trust is given to a large community of anonymous “miners”
• Anyone can be a miner (just download the software, or buy a specialized Bitcoin-mining rig)
• Miners are rewarded based on doing (a lot!!!) of computation
• Assumption: the set of honest miners will have more compute power than any set of dishonest miners
Bitcoin

- Public Ledger
  - Every transaction recorded and world-readable
  - “payer” and “payee” in a transaction are public keys, and you can use a different public key for each transaction
    - Public key X pays public key Y some amount
Format of Ledger: Blockchain
The Ledger

From transaction x8, X pays A 74.92 (hash=x15)
From transaction x11, Z pays B 38.22 (hash=x16)
From transaction x15, A pays C 74.21 (hash=x17)
From transaction x4, Q pays D 855.21 (hash=x18)
From transaction x17, C pays D 74.03 (hash=x19)
From transaction x18, D pays E 25.11, and F 830 (hash=x20)
etc.
The Ledger

From transaction x8, X pays A 74.92 (hash=x15)
From transaction x11, Z pays B 38.22 (hash=x16)
From transaction x15, A pays C 74.21 (hash=x17)
From transaction x4, Q pays D 855.21 (hash=x18)
From transaction x17, C pays D 74.03 (hash=x19)
From transaction x18, D pays E 25.11, and F 830 (hash=x20)
e etc.

Difference between received quantity and paid quantity is a transaction fee (a “tip” to miner who creates this block)
The Ledger – multiple outputs

<table>
<thead>
<tr>
<th>Transaction</th>
<th>Description</th>
<th>Hash</th>
</tr>
</thead>
<tbody>
<tr>
<td>x8</td>
<td>X pays A 74.92</td>
<td>x15</td>
</tr>
<tr>
<td>x11</td>
<td>Z pays B 38.22</td>
<td>x16</td>
</tr>
<tr>
<td>x15</td>
<td>A pays C 74.21</td>
<td>x17</td>
</tr>
<tr>
<td>x4</td>
<td>Q pays D 855.21</td>
<td>x18</td>
</tr>
<tr>
<td>x17</td>
<td>C pays D 74.03</td>
<td>x19</td>
</tr>
<tr>
<td>x18</td>
<td>D pays E 25.11, and F 830</td>
<td>x20</td>
</tr>
</tbody>
</table>

Note: Multiple outputs
- Lets you give yourself change
- Or you can split the amount among different entities
Anonymity

• Not really anonymous
• Entire world knows sequence of public keys
  – Merchant likely to know who belongs to the public key that paid him (he has to ship the merchandise, for instance)
  – Other instances where who owns a public key might be known
  – And then analyzing transaction sequence A pays B pays C can give information
Valid Transactions

• Suppose A pays B output of transaction with hash x
• How to tell if the transaction is valid?
  – Transaction (A pays B) must be validly signed by A
  – Need to find transaction with hash x, to make sure A owns that amount (A was the payee)
    • Presumably, can do some pre-processing so a hash=x can be found more efficiently than linear search
  – Need to search every transaction since, to make sure A hasn’t paid that transaction to someone else
    • Again, an implementation can optimize this by keeping track of unspent transactions
No central authority

• Central thing (like a bank) would make definitive ledger easy and efficient
• But Bitcoin is completely distributed, no pre-ordained trusted things
  • Where do Bitcoins come from?
  • How is the ledger agreed-upon?
Blockchain

• The blockchain is a set of blocks, and is the format of the ledger of all transactions

• A block is a set of new, valid transactions to add to the blockchain
  – Whose cryptographic hash is very very very hard to compute

• Bitcoin tries to cause a block to be added by the community of miners about every 10 minutes

• If blocks are found too quickly on average, hash difficulty is increased. If too slow, hash difficulty is decreased
Cryptographic Hashes

• A good hash is like a random number
  – As if, for every input, a random number were generated
• Probability that 1st bit = 0 for random input is 50%
• Probability that top 10 bits = 0 for random input is $1/2^{10}$
• The maximum winning hash value in Bitcoin is modified to make finding blocks easier or harder
• Currently the hash has to have 70 leading zeroes!
How to Find a Block with Hash with 70 leading zeroes

- Insert a bunch of valid pending transactions
- Choose a random number for the “nonce”
- Compute the hash
- If the hash doesn’t have 70 leading 0’s (probability of $1 - 1/2^{70}$ that it won’t), choose a different random number
- Repeat (about $2^{70}$ times)
How to Find a Block with Hash with 70 leading zeroes

- Insert a bunch of valid pending transactions
- Choose a random number for the “nonce”
- Compute the hash
- If the hash doesn’t have 70 leading 0’s (probability of $1 - 1/2^{70}$ that it won’t), choose a different random number
- Repeat (about $2^{70}$ times)
- If you’re lucky enough to be the first to find a next block with a small enough hash value, you get rewarded with some Bitcoins
Miner Reward

• If you are the first to find a valid next block, your reward:
  – Some number of Bitcoins (currently, 12.5), plus any transaction fees
What’s the purpose of all this hashing
What’s the purpose of all this hashing

• Estimated total: $10^{18}$ hashes per second!!!
What’s the purpose of all this hashing

- Estimated total: $10^{18}$ hashes per second!!!
- Remember, anyone can be a miner
- So “trust” is given because someone is willing to do that much worthless computation
What’s the purpose of all this hashing

• Estimated total: $10^{18}$ hashes per second!!!
• Remember, anyone can be a miner
• So “trust” is given because someone is willing to do that much worthless computation
• Assumption: there will be a lot of honest miners, and no dishonest miner (or collection of them) will be able to dominate the compute resources
Suppose a set of miners had most of the compute resources

- They could undo transactions and create an alternate history
- Including double-spending their own money, after the payee thought it was safely paid
- Basically, they could bring down the whole Bitcoin concept
Broadcast

- This technology is also expensive in network bandwidth
  - A transaction has to be broadcast to, at least, most of the miners
  - When a miner finds a hash to create a new block, that block needs to be broadcast to most of the miners
  - If your transaction was not included in the winning block, hopefully it will be in a later block
  - Basically, a gossip protocol: your node somehow knows some others, they tell others, etc.
If this node dies, partition
Blockchain forks

• It’s possible for multiple miners to semi-simultaneously find a hash
• Which means the blockchain will fork
• If a miner sees multiple ones, it works on the longest valid one (most blocks, and all contained transactions valid)
• So hopefully the network will converge on a single chain
Blockchain forks

- It’s possible for multiple miners to semi-simultaneously find a hash
- Which means the blockchain will fork
- If a miner sees multiple ones, it works on the longest valid one (most blocks, and all contained transactions valid)
- So hopefully the network will converge on a single chain
- This can take several blocks, so the advice is not to consider your transaction complete until it’s in block n, and the chain is n+6
  - Note: 6 blocks is about an hour!
Blockchain forks

- It’s possible for multiple miners to semi-simultaneously find a hash
- Which means the blockchain will fork
- If a miner sees multiple ones, it works on the longest valid one (most blocks, and all contained transactions valid)
- So hopefully the network will converge on a single chain
- This can take several blocks, so the advice is not to consider your transaction complete until it’s in block n, and the chain is n+6
  - Note: 6 blocks is about an hour!
- Until your transaction where A paid you is safely recorded, A can double spend, and cheat you
Blockchain forks

• If Internet were partitioned, or gossip network partitioned, miners on each side would happily be adding blocks to the blockchain

• When the partitions rejoined, whichever side had the longest blockchain would win…everything beyond where they forked, in the losing blockchain, is no longer in the ledger
  – any Bitcoins mined aren’t there any more
  – any transactions in the losing fork are no longer in the ledger
  – any payers in transactions in losing fork can re-spend
Really bad forks

- March 2013: a new version of Bitcoin software had some “harmless” tweaks, like making the code more efficient
- But there was a block that couldn’t be processed by the previous version
- So, permanent fork!
- Eventually people noticed (hours and many blocks later), and downgraded to the older version
- But what if lots of different implementations, and no way of contacting all the miners?
Finite Total Bitcoins

• Initially the reward for finding the winning hash of a block was 50 Bitcoins
• Then it was halved to 25. Recently (July 2016), halved again to 12.5
• Every 4 years (or so), the reward (# of Bitcoins) halved
• Eventually, (about 2140) the reward will be zero
  – but it will be negligible long before that
  – miner still gets transaction fees
• As of Dec 2015, about 71% of Bitcoins have been mined
Transaction Fees

- Bitcoin claims it has lower transaction fees than, say, credit cards
- And today that’s true…most transactions do not include a transaction fee, and yet they get swept into the blockchain
- But if too many transactions per time to fit into the block, miners will only include transactions with highest transaction fee
- Blocks are finite size. Each transaction is work for the miner (to ensure transaction valid, and more bytes in block means more hashing work)
- And as reward depends on fees rather than mined coins, transaction fees will escalate – currently with mining reward, miner gets about $5.85 reward per transaction in block (today, transaction fees only account for about 3% of reward)
- Note: The open source client has some algorithm for fees
Interesting statistics

• https://blockchain.info/stats
• http://www.coindesk.com/data/bitcoin/
Full vs Lightweight Nodes

- Imagine if your smartphone needed to store the entire blockchain!
- As well as search through the whole blockchain to see if the payer really owns the money he’s paying you
- So instead, a (lightweight) client node is configured with contact information for a few trusted full nodes, which will check transactions
Full vs Lightweight Nodes

• Imagine if your smartphone needed to store the entire blockchain!
• As well as search through the whole blockchain to see if the payer really owns the money he’s paying you
• So instead, a (lightweight) client node is configured with contact information for a few trusted full nodes, which will check transactions
• Note the irony: Supposedly nothing needs to be trusted, but clients have to trust the configured full nodes
How much energy?

• Assumptions
  – If mining very lucrative, more miners will join
  – If cost of electricity is more than reward, miners will drop out
  – So, amount spent on electricity will be a little less than reward
How much energy?

- Hard to know exactly – depends on a lot of things, but let’s try
  - Miner reward over the last year has been about a million dollars a day
  - Reports are that miner reward barely covers the cost of electricity
  - At 10 cents per Kilowatt-hour, this is about enough electricity for 200,000 American homes
  - Or about ½ a nuclear power plant
But

- Psychology: if you buy a mining rig, you want to use it, even if paying more for electricity than you’re reaping
- You may not know how much the Bitcoin mining is costing you
- There are countries where electricity is highly subsidized, so mining there is way cheaper
- You may be stealing the electricity, so you might not care
  - Bots, invalid public cloud accounts
  - Harder to win against specialized mining machines, but hey, if the electricity is “free”
How much storage?

• About 85 GB, up about 40 GB since last year
  – On each full node (about 5000 nodes currently)
• Can grow at most 50GB per year, with fixed size blocks and 10 minutes/block
• But they’re talking about increasing block size
  • https://blockchain.info/charts/blocks-size
• And this is stored at every full node
It’s hard to protect your Bitcoins

• If you lose your wallet, all your money is gone
• If your computer is broken into, your money can be stolen and spent by someone else
• With banks, there is some legal protection of user money
  – I assume…what if you don’t protect your password? Do you get stolen money back?
  – It’s probably somewhat possible to trace where money went, but maybe foreign banks provide cover
Why I’m not a fan

• Enormously wasteful of energy, bandwidth, storage
  – Imagine if it were more popular, and used for more things!
• Transaction fees will eventually need to be really large
• Fragile, especially if any diversity in implementations (permanent forks)
• Trust model (nation state or mining pool could have too much compute and
double-spend, refuse to record transactions they don’t like, etc.)
• Difficult for user to protect money
• Does coffee shop have to make you wait for an hour to make sure that they’ll
get paid?
• Permanent record of every transaction
• Limited # of transactions per unit time (blocks are finite size, one block every
fixed amount of time)
What do people think are the good things about Bitcoin/blockchain?

• “Distributed ledger”
  – I claim distributed databases are known technology
• “No central authority”
  – I claim you can have several trusted authorities, with a transaction being valid only if more than half have signed…much more efficient, and sensible than thousands of anonymous miners
• “No greedy banks taking 3% transaction fees”
  – I claim Bitcoin transaction fees will be even worse
• “Immutable, permanent record”
  – Can do that with signatures, and storage in multiple places
• “Proof that event x occurred at some time”
  – Can use public trusted time-stamping services
Confusion

• People are calling all sorts of things “blockchain”
• They might be totally reasonable technology…but they aren’t “revolutionary new technology”, and they aren’t Bitcoin’s blockchain
Thank you!