Bitcoin, Security for Cloud & Big Data

CS 161: Computer Security

Prof. David Wagner

April 18, 2013

Bitcoin

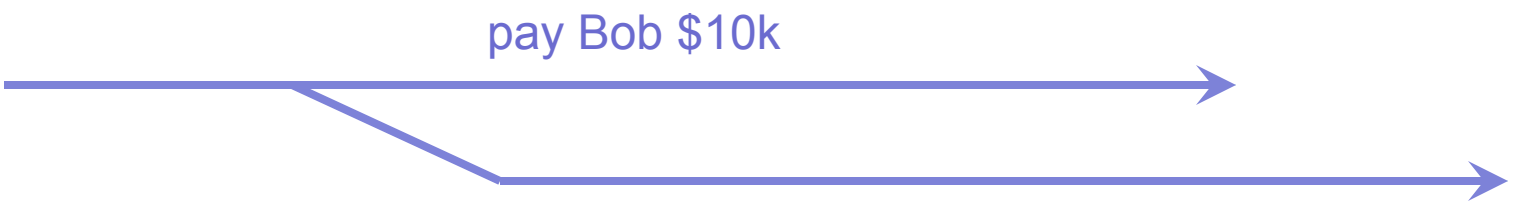
- Public, distributed, peer-to-peer, hash-chained audit log of all transactions ("block chain").
- Mining: Each entry in block chain must come with a proof of work (its hash value ends in *k* zeros). Thus, appending takes computation.
- Lottery: First to successfully append to block chain gets a small reward (if append is accepted by others). This creates new money. Each block contains a list of transactions, and identify of miner (who receives the reward).
- Consensus: If there are multiple versions of the block chain, longest one wins.

Bitcoin

- Transactions: If Alice wants to give $10 to Bob, she signs this transaction. She gives the signed transaction to all miners and asks them to include it in the block they're trying to append to the chain.
- Honest miners check integrity of block chain entries and try to append to the latest, longest valid version of block chain.
- Bob knows he has received $10 once this transaction appears in the consensus block chain.

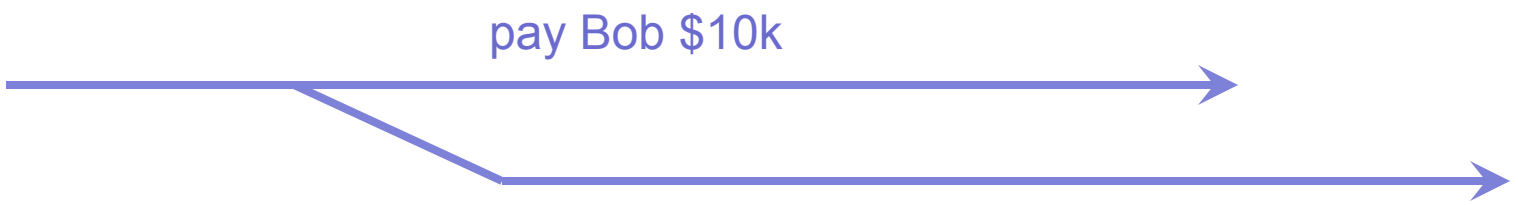
Consensus

- Can Mallory fork the block chain?
- Say she buys Bob's from him for $10,000 in Bitcoins. Then, she goes back in time and, starting from the block chain just before this transaction was added to it, she starts appending new entries from there. Can she get others to accept this forked chain, so she gets her $10,000 back?



Consensus

- Can Mallory fork the block chain?
- Answer: No, not unless she has ≥51% of the computing power in the world. Longest chain wins, and her forked one will be shorter (unless she can mine new entries faster than aggregate mining power of everyone else in the world).



How Bitcoin Addresses Criticisms of Naïve Scheme

- *Initial balance is arbitrary*: in Bitcoin, initial balances are zero
- *Broadcasting is expensive and doesn't scale:* gossip protocol
- *A conspiracy of n/2 malicious computers can fork the audit log and steal all the money:* they'd have to own 51% of all the computing power in the Bitcoin world
- *Sybil attacks: Anyone can set up millions of servers and thus have a 50% majority:* they'd have to own 51% of all the computing power in the Bitcoin world

Discussion

- How can Alice turn dollars into bitcoins, or vice versa?
- Is Bitcoin anonymous?
- Should I think of Bitcoin as a short-term currency or as a long-term investment?
- Is it ethical to build a system that relies upon wasting CPU cycles (and thus energy)?

BitCoin Take-away

- Crypto tools allow for sophisticated solutions to integrity and trust in peer-to-peer systems

Big Data in the Cloud

Trends in computing:

- “Big data”: Easy to collect lots and lots of data about us
- “Cloud computing”: Cheaper to store data in the cloud, and do computation there

What are the security and privacy implications of these trends?

Big Data in the Cloud

Trends in computing:

- “Big data”: Easy to collect lots and lots of data about us
- “Cloud computing”: Cheaper to store data in the cloud, and do computation there

What are the security and privacy implications of these trends?

- Privacy – companies know a lot about us
- Data security – a security breach exposes all our data

Potential Solutions

Some possible ways to mitigate the threat:

- Policy: Minimize data collection or retention, limit who can access stored data or for what purposes
- Technology: Encrypt data while it is stored on cloud servers

Potential Solutions

Some possible ways to mitigate the threat:

- Policy: Minimize data collection or retention, limit who can access stored data or for what purposes
- Technology: Encrypt data while it is stored on cloud servers – *but then how can they do any useful computation on our data?*

Case Study: DropBox

- DropBox lets you store your files in the cloud.
- For efficiency reasons, if Alice and Bob have the same file (pharrell_williams_happy.mp3), DropBox doesn't want to store it twice.
- For security reasons, it'd be nice if DropBox could encrypt your files on your computer (with a key only you know) and upload the encrypted version.
- But if we encrypt the same file twice, we get two different ciphertexts.

Case Study: DropBox

- DropBox lets you store your files in the cloud.
- For efficiency reasons, if Alice and Bob have the same file (pharrell_williams_happy.mp3), DropBox doesn't want to store it twice.
- For security reasons, it'd be nice if DropBox could encrypt your files on your computer (with a key only you know) and upload the encrypted version.
- But if we encrypt the same file twice, we get two different ciphertexts.
- *How does DropBox do de-duplication on encrypted files?*

DropBox's Solution

- What DropBox actually does:
  - To upload file *x*, send SHA256(*x*) to DropBox. If it's not a duplicate, send *x* to DropBox over SSL, and they'll encrypt it using a fixed key *k* (same for all users) and store $E_k(x)$ on their servers.
- Problems:
  - DropBox has *k* so can decrypt all your data.
  - A bug or security breach in DropBox can expose all your data.
  - In fact, on 6/19/2011, DropBox *did* have a bad bug, where they screwed up the authentication: you could log into someone else's account without knowing their password (just enter any password, and you're in). This exposed *everyone's* files to the world.

Better Solution for DropBox?

- A natural attempt at a better solution:
  To upload file *x*, send to DropBox
  $E_k(x)$, SHA256(x)
  where *k* is your personal key.
- This *does* let DropBox detect duplicates, but it has a problem.
  - Say Alice uploads file *x* (encrypted under her key k_1), then Bob uploads the same file *x* (encrypted under his key k_2). DropBox can detect it's a duplicate, since it has the same hash. However, the copy on DropBox's servers is encrypted under Alice's key k_1, so Bob won't later be able to decrypt.

A Better Solution for DropBox

- Better solution: "Convergent" encryption. Upload
  $C = \text{AES-CBC}_k(x)$,
  where $k = \text{SHA256}(x)$ and IV = 0. Also upload an encryption of k under your personal key.
- Now encrypting the same file twice gives the same ciphertext C, so C only needs to be stored once.
  - C is a deterministic function of x, so if Alice and Bob upload the same file x, DropBox only has to store C once. DropBox does store encryption of k under Alice's key and under Bob's key, but both of those are short.
- This is what DropBox *should* have done.

- *What's the potential weakness of this?*

Case Study: Encrypted Email

- My email is stored in the cloud on a server.
- For security reasons, I want it to be stored in encrypted form, so I don’t have to trust the server.
- But I also want to be able to do keyword search on all my email.

Case Study: Encrypted Email

- My email is stored in the cloud on a server.
- For security reasons, I want it to be stored in encrypted form, so I don't have to trust the server.
- But I also want to be able to do keyword search on all my email.
- *How can I search on encrypted email?*

Solution for Encrypted Email

- One solution: Each word *w* is encrypted separately and deterministically:
  $E_k(w) = \text{AES-CBC}_k(w)$ where IV = SHA256(*w*)
- Advantage: Keyword searches just work, as long as I encrypt the keyword I'm searching on.
  Problem: This leaks a lot of data about my email.

Solution for Encrypted Email

- One solution: Each word *w* is encrypted separately and deterministically:
  $E_k(w)$ = AES-CBC$_k(w)$ where IV = SHA256(*w*)
- Advantage: Keyword searches just work, as long as I encrypt the keyword I'm searching on.
  Problem: This leaks a lot of data about my email.
- More secure solution: For each word *w*, store
  r, SHA256(*r*, $E_k(w)$)
  where r is random and different each time, and $E_k(w)$ is deterministic encryption as above.
- To search for word *w*, send $x = E_k(w)$ to server. For each *r*, *y* on the server, server can test whether SHA256(*r*, *x*)=*y*.

Match-making

- Alice and Bob are cryptographers and want to find out if they're interested in each other romantically, but neither wants to suffer possible rejection.
- Can we build a match-making service where they both get notified if they're both interested in each other, but otherwise they learn nothing?

Match-making

- Alice and Bob are cryptographers and want to find out if they're interested in each other romantically, but neither wants to suffer possible rejection.
- Can we build a match-making service where they both get notified if they're both interested in each other, but otherwise they learn nothing?
- Solution: Use a trusted server S. Alice sends x to S, where $x = 1$ if she is interested in Bob or 0 if not. Bob sends y to S. S computes $z = x \wedge y$ and sends z to both Alice and Bob.

Match-making

- Alice and Bob are cryptographers and want to find out if they're interested in each other romantically, but neither wants to suffer possible rejection.
- Can we build a match-making service where they both get notified if they're both interested in each other, but otherwise they learn nothing?
- Solution: Use a trusted server S. Alice sends x to S, where $x = 1$ if she is interested in Bob or 0 if not. Bob sends y to S. S computes $z = x \wedge y$ and sends z to both Alice and Bob.
- *Can Alice and Bob do this on their own without trusting any server?*