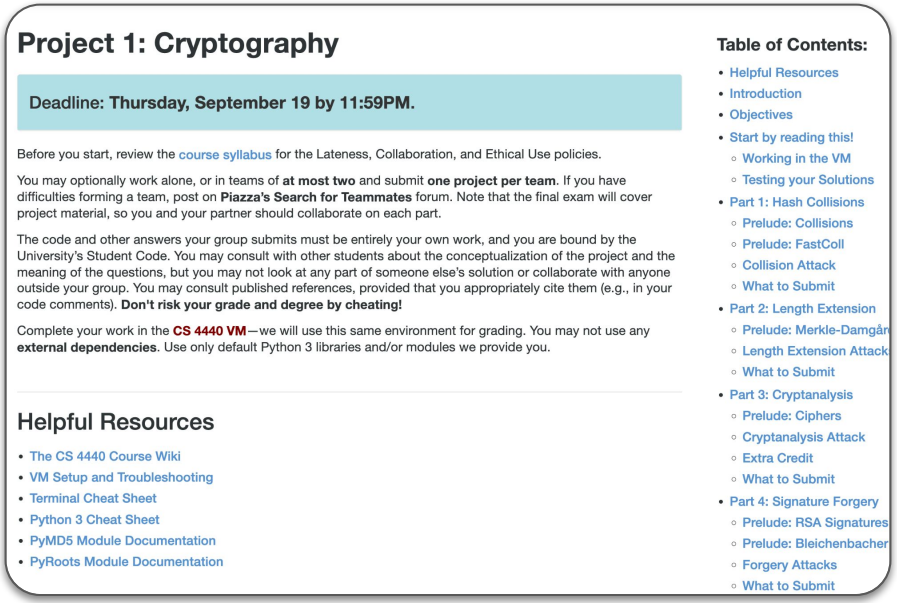
Week 4: Lecture B

Security in Practice: Cryptocurrency

Thursday, September 12, 2024

Announcements

- **Project 1: Crypto** released (see **Assignments** page on course website)
  - **Deadline:** Thursday, September 19th by 11:59 PM



Project 1: Cryptography

Deadline: Thursday, September 19 by 11:59PM.

Before you start, review the course syllabus for the Lateness, Collaboration, and Ethical Use policies.

You may optionally work alone, or in teams of **at most two** and submit **one project per team**. If you have difficulties forming a team, post on **Piazza's Search for Teammates** forum. Note that the final exam will cover project material, so you and your partner should collaborate on each part.

The code and other answers your group submits must be entirely your own work, and you are bound by the University's Student Code. You may consult with other students about the conceptualization of the project and the meaning of the questions, but you may not look at any part of someone else's solution or collaborate with anyone outside your group. You may consult published references, provided that you appropriately cite them (e.g., in your code comments). **Don't risk your grade and degree by cheating!**

Complete your work in the **CS 4440 VM**—we will use this same environment for grading. You may not use any **external dependencies**. Use only default Python 3 libraries and/or modules we provide you.

Helpful Resources

- The CS 4440 Course Wiki
- VM Setup and Troubleshooting
- Terminal Cheat Sheet
- Python 3 Cheat Sheet
- PyMD5 Module Documentation
- PyRoots Module Documentation

Table of Contents:

Progress on Project 1

Finished everything!

0%

Finished Parts 1 – 3

0%

Finished Parts 1 – 2

0%

Finished Part 1

0%

Haven't started :(

0%

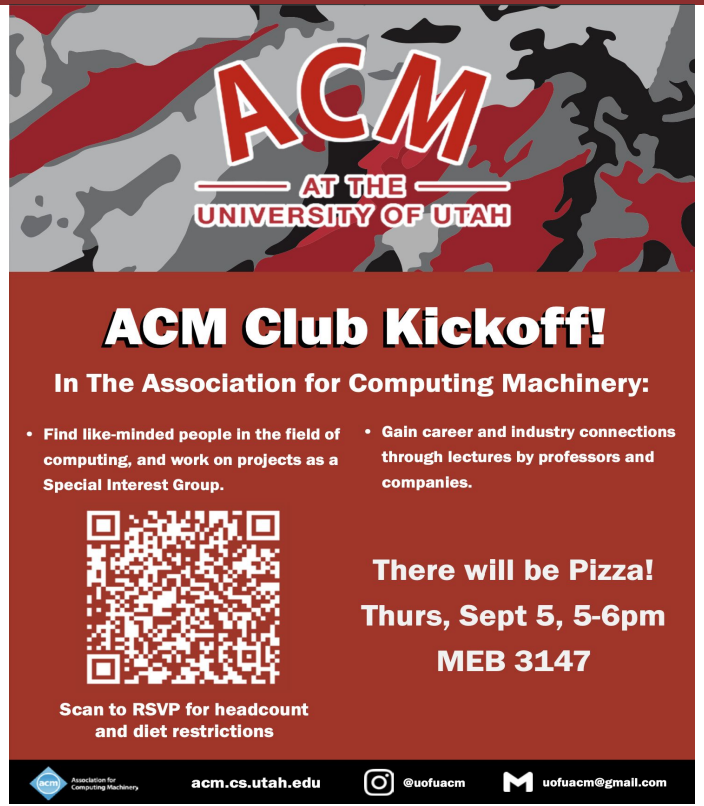
Announcements

utahsec

See Discord for meeting info!

utahsec.cs.utah.edu

Announcements



Questions?

Last time on CS 4440...

Key Exchange
Digital Signatures
RSA
Bleichenbacher's Attack
Key Management Rules

Asymmetric vs. Symmetric Crypto

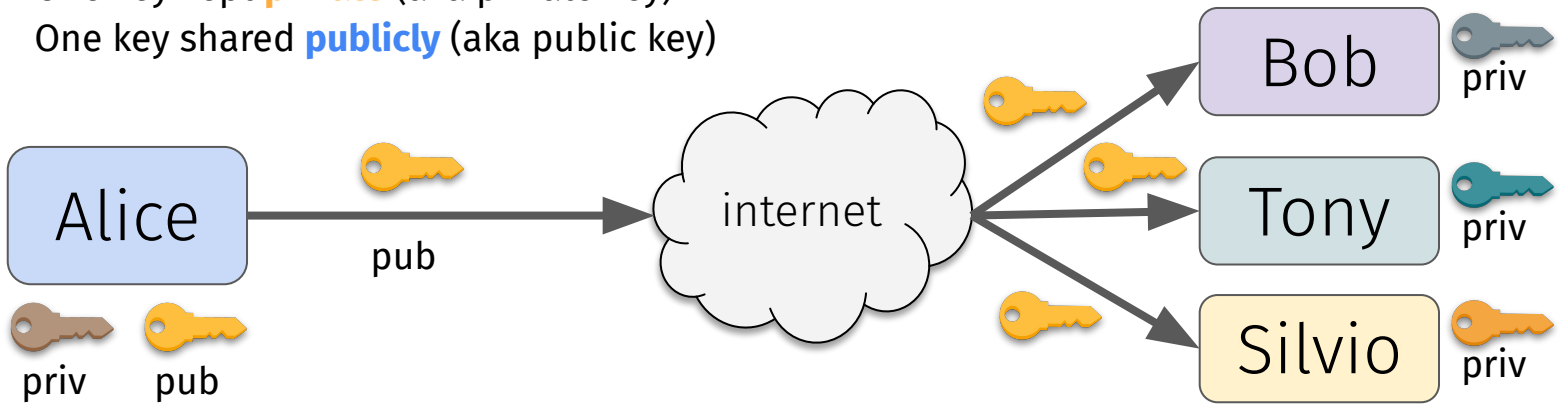
Symmetric Crypto

Asymmetric Crypto

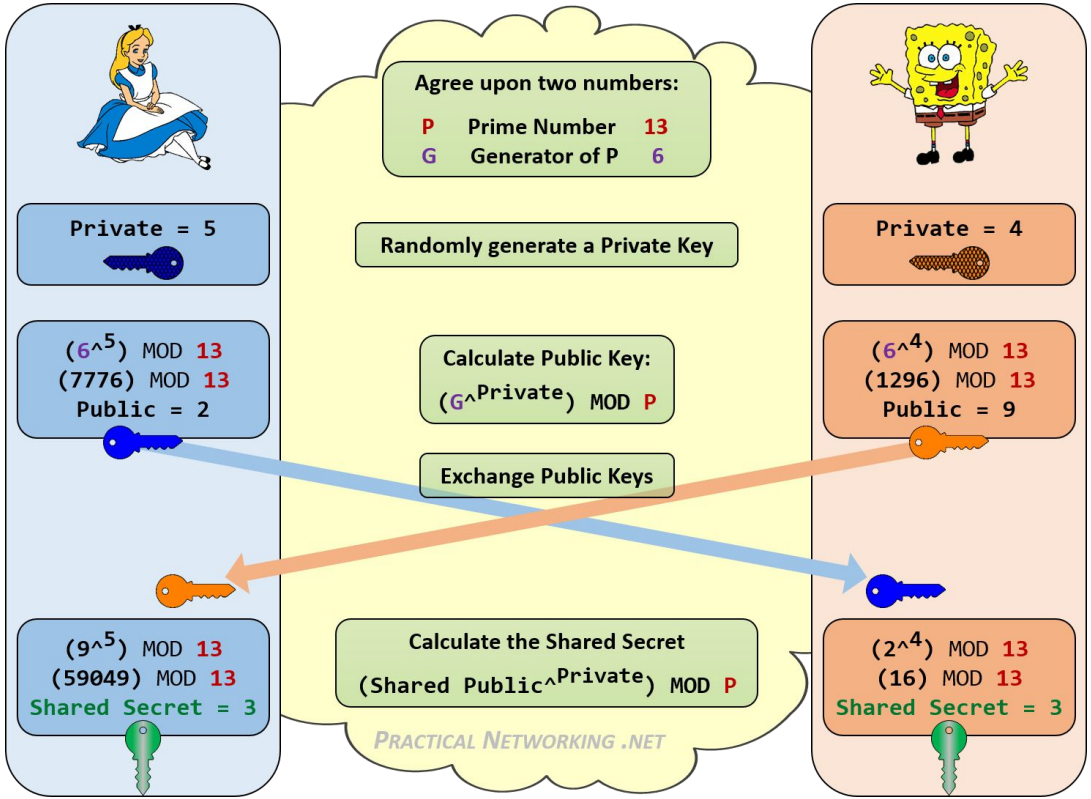
Asymmetric Encryption (aka “Public Key”)

- **Key idea:** want a **asymmetric** approach to find a symmetric key
  - Don’t want to have to pre-share keys in advance

- Suppose users can have **two keys:** encryption and decryption
  - Keys generated in pairs using well-understood mathematical relationship
  - One key kept **private** (aka private key)
  - One key shared **publicly** (aka public key)

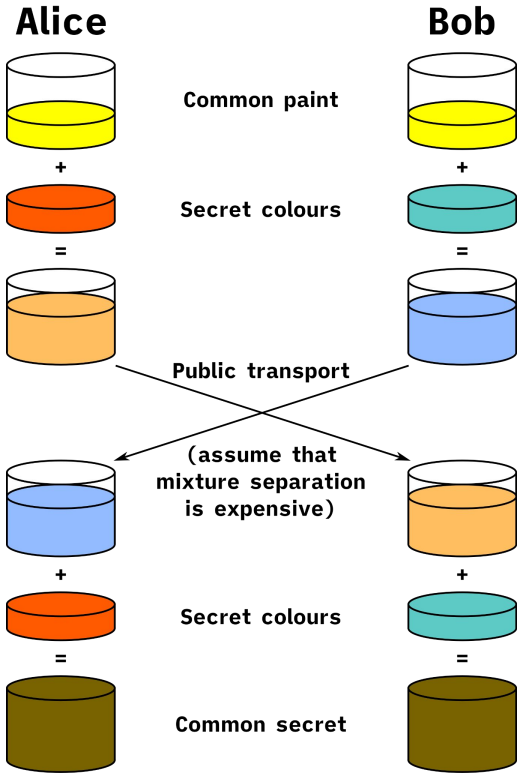


Diffie-Hellman Key Exchange



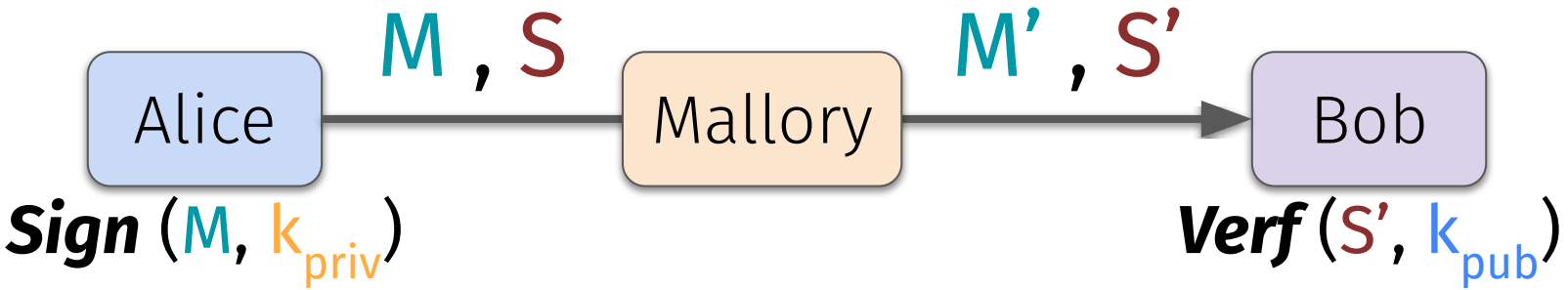
A visual analogy of Diffie-Hellman

- Diffie-Hellman's exponentiation
  - Think of it like mixing different paint colors
- Hard to **invert** to original colors? **Yes!**
- Two different ways of arriving to the same final result (i.e., the shared key)
  - Done as a **"public conversation"**



Authenticity via Digital Signatures

- **Key generation:** Alice generates key pair: k_{pub} (public) and k_{priv} (private)
- Alice **signs** message **M** with k_{priv} resulting in signature **S** = ***Sign*** (**M**, k_{priv})
- **Anyone** possessing Alice's k_{pub} can **check** signature via ***Verf*** (***S'***, k_{pub})
  - If received message and signature **verified**, then message is **authentic**—from Alice!



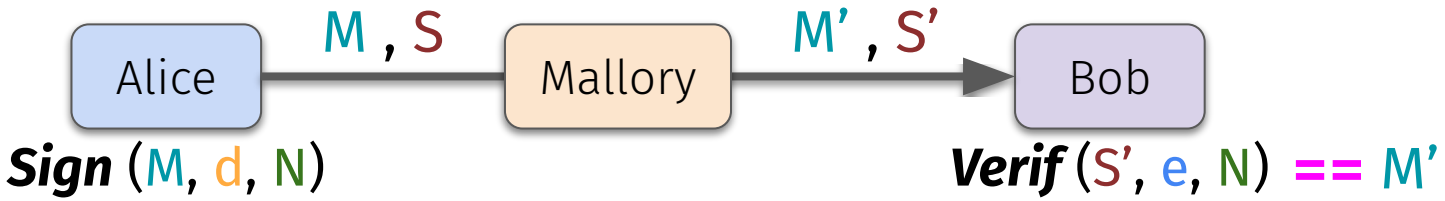
Authenticity via Digital Signatures

Unforgeability: computationally **infeasible** for **Mallory** to guess **S** or Alice's k_{priv}

... even if **Mallory knows** Alice's k_{pub} or other signatures from other messages!

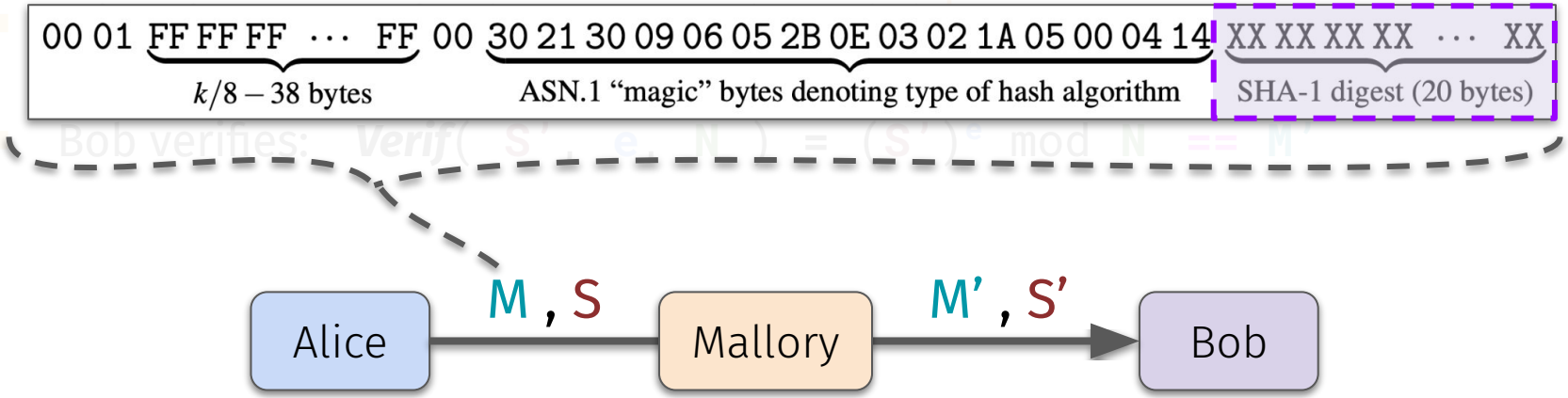
RSA Digital Signatures

- **Public key** = (e, N) where e is **relatively prime** to (p-1)(q-1)
- **Private key** = (d, N) where (e*d) **mod** ((p-1)(q-1)) = 1
- Alice signs: S = ***Sign***(M, d, N) = $(M)^d$ mod N
- Bob verifies: ***Verif***(S', e, N) = $(S')^e$ mod N == M'



RSA Digital Signatures

RSA Messages are really this giant-long integer construction; SHA-1 digest = **SHA-1(plaintext)**



RSA Digital Signatures

Bob checks that $(S')^e \bmod N == M'$

Mallory can **forge Alice's messages**, but can't forge **her signatures**—why?

RSA Digital Signatures

Bob checks that $(S')^e \bmod N == M'$

Mallory can **forge Alice's messages**, but can't forge **her signatures**—why?

Because Alice **signs** via **her *private* key**!

RSA vs. Diffie-Hellman

- **Diffie-Hellman:** a protocol for secure **key exchange**
  - Idea of a "**public conversation** to derive a **shared secret key**"
  - Hardness assumption based on **discrete log problem**
    - Given $\mathbf{g^x}$ **mod p**, find the exponent **x**
    - Really hard if **p** is a **large prime** number!

RSA vs. Diffie-Hellman

- **Diffie-Hellman:** a protocol for secure **key exchange**
  - Idea of a "**public conversation** to derive a **shared secret key**"
  - Hardness assumption based on **discrete log problem**
    - Given $\mathbf{g^x}$ **mod p**, find the exponent **x**
    - Really hard if **p** is a **large prime** number!

- **RSA:** a **cryptosystem**; can use for encryption, signing
  - Based on **principles of Diffie-Hellman** ("public" key derivation)
  - Hardness assumption based on **integer factorization problem**
    - Given **N**, find two integers such that **x * y = N**
    - Really hard if **x** and **y** are **large prime** numbers!

RSA vs. Diffie-Hellman

- **Diffie-Hellman:** a protocol for secure **key exchange**
  - Idea of a "**public conversation** to derive a **shared secret key**"
  - Hardness assumption based on **discrete log problem**
    - Given g^x **mod** p, find the exponent x
    - Reall
- **RSA:** a **crypto**
  - Based on p
  - Hardness as
    - Given **N**, find two integers such that **x * y = N**
    - Really hard if **x** and **y** are **large prime** numbers!

Security of both hinges on difficulty of **large prime numbers**!

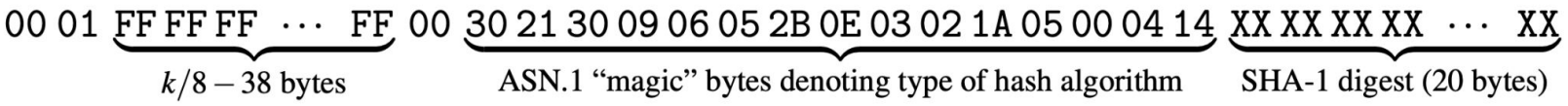
Bleichenbacher Attack: Forging RSA Digital Signatures

- Check if **message** == (**signature**)$^{\text{exponent}}$ modulo (**N**)
  - In this problem, we know **message** and want to find **signature**
- Recall **N** computed by multiplying two huge prime numbers
  - **Mallory has zero hope of figuring these factors out** (integer factorization problem)
- Check if **message** == (**signature**)$^{\text{exponent}}$ modulo (**HugeUnfactorableNumber**)
- If **exponent** is **small**, what happens?

Bleichenbacher Attack: Forging RSA Digital Signatures

- Check if **message** == (**signature**)$^{\text{exponent}}$ modulo (**N**)
  - In this problem, we know **message** and want to find **signature**
- Recall **N** computed by multiplying two huge prime numbers
  - **Mallory has zero hope of figuring these factors out** (integer factorization problem)
- Check if **message** == (**signature**)$^{\text{exponent}}$ ~~modulo (**HugeUnfactorableNumber**)~~
- If **exponent** is **small**, what happens? Right-hand modulo expression is **null**
  - With **message** in-hand, **Mallory can retrieve** the **signature**!

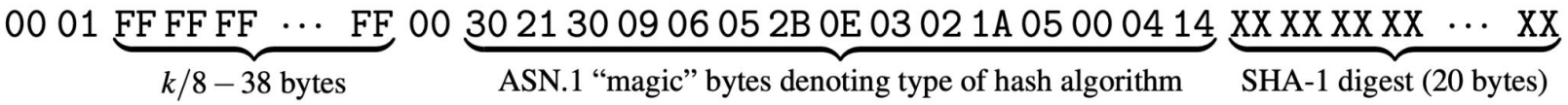
Bleichenbacher Attack: Forging RSA Digital Signatures



SHA1("Go Chiefs!")

- Assume key is **2048 bits long**
- Prefix **FF**'s must be **((2048/8)—38)** bytes
  - = **218** total **FF**'s
- Where does **38** come from?

Bleichenbacher Attack: Forging RSA Digital Signatures

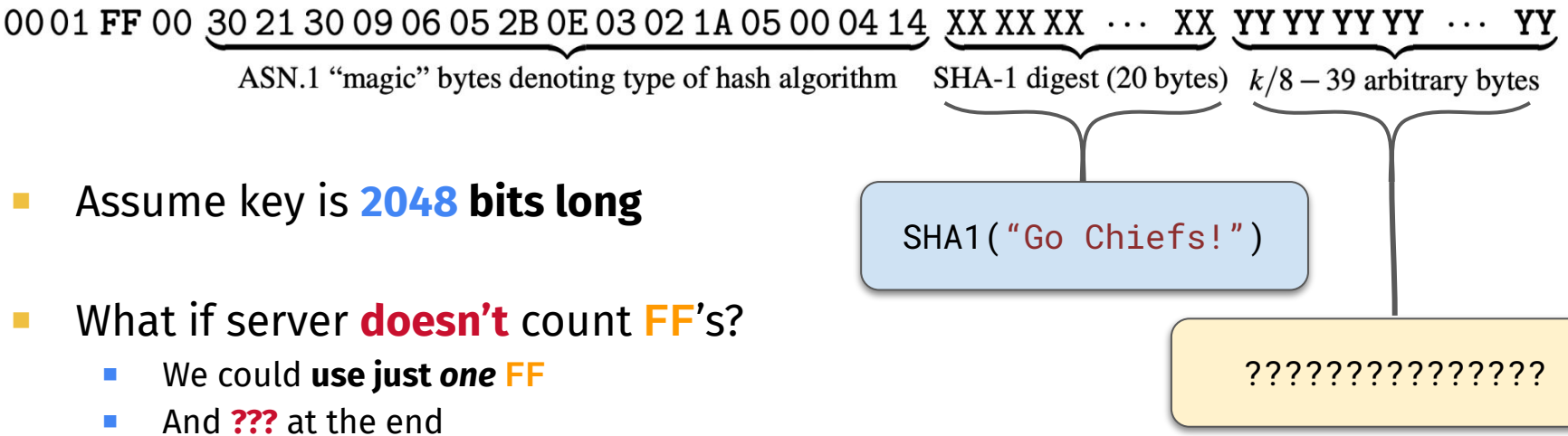


SHA1("Go Chiefs!")

- Assume key is **2048 bits long**
- Prefix **FF**'s must be **((2048/8)—38)** bytes
  - = **218** total **FF**'s
- Where does **38** come from?
  - **20-byte** SHA-1 digest
  - **15-byte** ASN.1 hash specifier
  - **3 more bytes** (`00`, `01`, `00`)

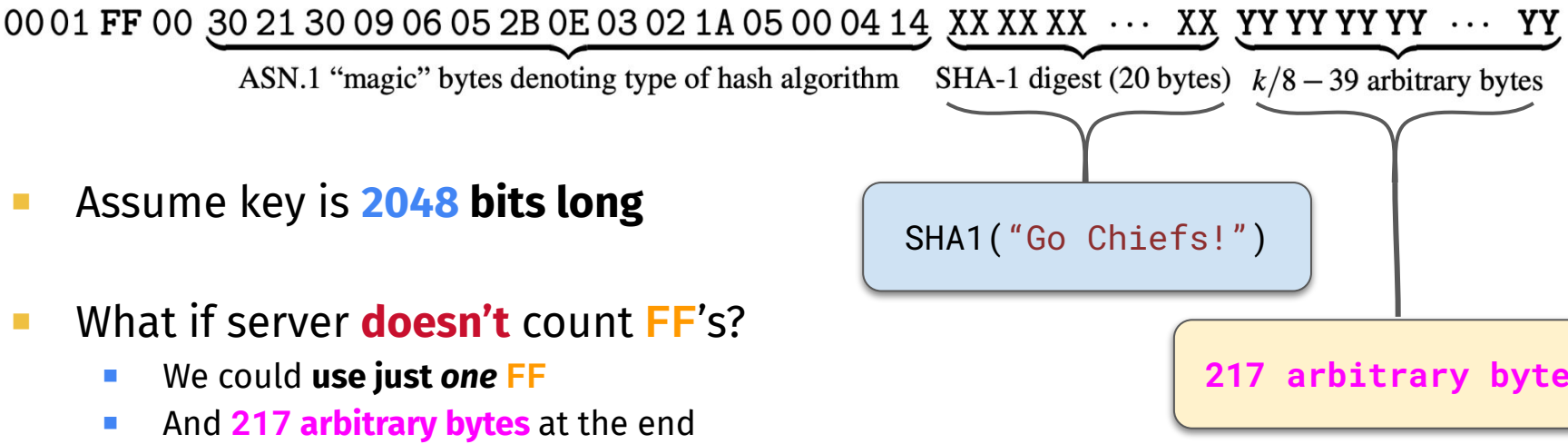
If number of **FF**'s don't match **218**, reject message!

Bleichenbacher Attack: Forging RSA Digital Signatures



- Assume key is **2048 bits long**
- What if server **doesn't** count **FF**'s?
  - We could **use just *one* FF**
  - And **???** at the end

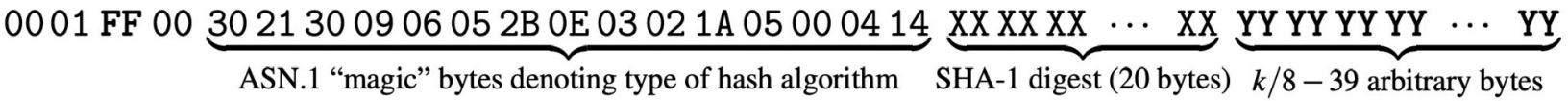
Bleichenbacher Attack: Forging RSA Digital Signatures



- Assume key is **2048 bits long**
- What if server **doesn't** count **FF**'s?
  - We could **use just *one* FF**
  - And **217 arbitrary bytes** at the end
    - These end up **not being checked!**

Bleichenbacher Attack: Forging RSA Digital Signatures

- How about ***N*th-rooting** the **insecure** message construction?



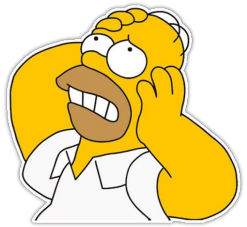
  - It is highly unlikely that you get a **perfect root!**
  - Your signature has to be an integer—no decimal remainder!
    - Thus, **message will not equal** (**signature**)$^{\text{exponent}}$
    - **Attack fails!**

Bleichenbacher Attack: Forging RSA Digital Signatures

Suppose **Mallory's** $M_{evil} = 300$
and the **server's exponent** $= 3$

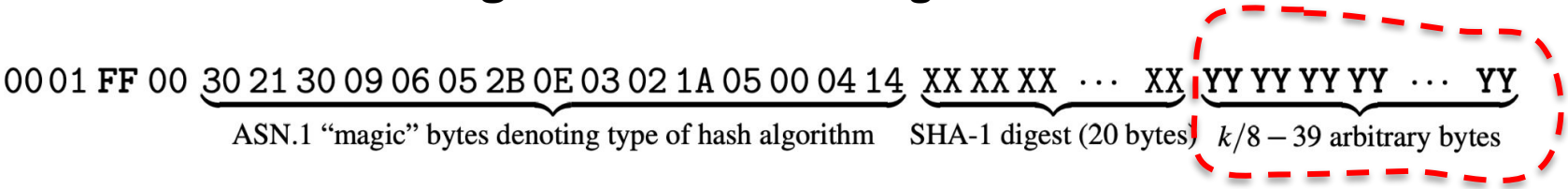
Mallory computes $S_{evil} = M_{evil}^{1/3} = 6.694$

Server checks signature: $6.000^3 \; != \; 300$



Bleichenbacher Attack: Forging RSA Digital Signatures

- How about ***N*th-rooting** the **insecure** message construction?



  - It is highly unlikely that you get a **perfect root!**
  - Your signature has to be an integer—no decimal remainder!
    - Thus, **message will not equal** (**signature**)$^{\text{exponent}}$
    - **Attack fails!**

  - But… we know that the **last 217 bytes of the message aren't checked** by the server!

Bleichenbacher Attack: Forging RSA Digital Signatures

- **Visualization of unchecked bytes:** compare 300 and 343 side-by-side:

```
1 0 0 1 0 1 1 0 0      (bytes 3–9 don't match!)
Perfect cube: 1 0 1 0 1 0 1 1 1
```

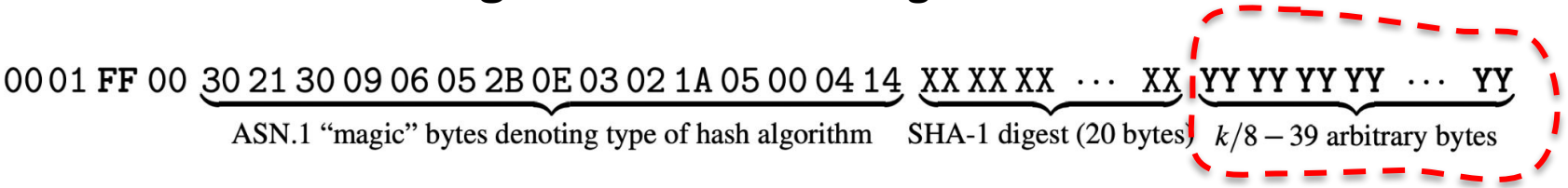
- Pretend that **everything after the first two bytes is ignored** by the server

```
1 0 0 1 0 1 1 0 0      (only care about bytes 1–2)
Perfect cube: 1 0 1 0 1 0 1 1 1
```

- **Success! Check passes**

Bleichenbacher Attack: Forging RSA Digital Signatures

- How about ***N*th-rooting** the **insecure** message construction?



  - It is highly unlikely that you get a **perfect root!**
  - Your signature has to be an integer—no decimal remainder!
    - Thus, **message will not equal** (**signature**)$^{\text{exponent}}$
    - **Attack fails!**

  - But… we know that the **last 217 bytes of the message aren't checked** by the server!
    - Thus, **we can "tweak" our signature** such that **message** == (**signature**)$^{\text{exponent}}$
    - When server computes (**signature**)$^{\text{exponent}}$, will get slightly different **message**—**that's ok!**

Bleichenbacher Attack: Forging RSA Digital Signatures

- How about ***N*th-rooting** the **insecure** message construction?

Small exponent + **insecure padding** enables Mallory to **forge signatures**... ***without*** knowing Alice's **private key**!

- It is h
- Your s
  -
  -
- But... we know that the last 217 bytes of the message aren't checked by the server!
  - Thus, **we can "tweak" our signature** such that **message** == (**signature**)^{exponent}
  - When server computes (**signature**)^{exponent}, will get slightly different **message**—**that's ok!**

RSA for Confidentiality and Integrity

- Subtle fact: RSA can also be used for **integrity** and **confidentiality**

- RSA for **integrity**:
  - **Goal:** Prove that message wasn't tampered
  - Encrypt ("**sign**") with **???**
  - Decrypt ("**verify**") with **???**

RSA for Confidentiality and Integrity

- Subtle fact: RSA can also be used for **integrity** and **confidentiality**

- RSA for **integrity**:
  - **Goal:** Prove that message wasn't tampered
  - Encrypt ("**sign**") with sender's **private key**
  - Decrypt ("**verify**") with sender's **public key**

- RSA for **confidentiality**:
  - **Goal:** Allow only intended recipient to read
  - **Encrypt** with **???**
  - **Decrypt** with **???**

RSA for Confidentiality and Integrity

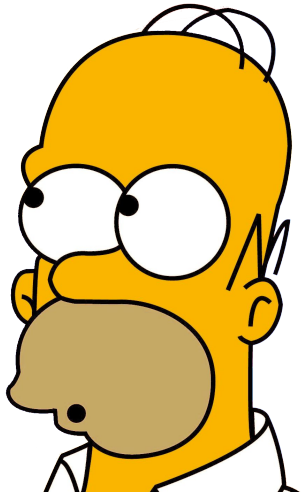
- Subtle fact: RSA can also be used for **integrity** and **confidentiality**

- RSA for **integrity**:
  - **Goal:** Prove that message wasn't tampered
  - Encrypt ("**sign**") with sender's **private key**
  - Decrypt ("**verify**") with sender's **public key**

- RSA for **confidentiality**:
  - **Goal:** Allow only intended recipient to read
  - **Encrypt** with recipient's **public key**
  - **Decrypt** with recipient's **private key**

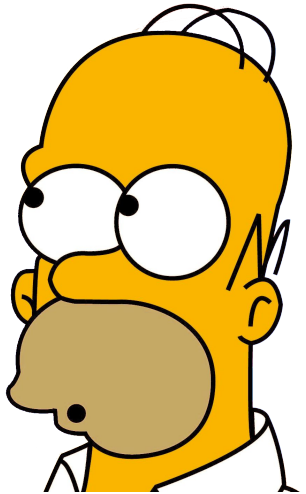
True or False: RSA vs. AES

- RSA is no more complex than AES
- RSA requires the same size keys as AES
- RSA requires less computation than AES
- RSA requires pre-sharing of keys



True or False: RSA vs. AES

- RSA is no more complex than AES
  - **False:** it's **far more complex**
- RSA requires the same size keys as AES
  - **False:** needs **much larger** keys (e.g., 10x larger)
- RSA requires less computation than AES
  - **False:** it's **1000x slower** than AES
- RSA requires pre-sharing of keys
  - **False:** it's **asymmetric**—that's why we love it!



True or False: Key Management

- Keys should have only one purpose
- It's okay to reuse the same key over and over again
- Digital storage is as safe as hardware storage
- Alice → Bob can use the same key as Bob → Alice

True or False: Key Management

- Keys should have only one purpose
  - **True:** **one key** for integrity, **one** for confidentiality, etc.
- It's okay to reuse the same key over and over again
  - **False:** keys become **more vulnerable** with time, reuse!
- Digital storage is as safe as hardware storage
  - **False:** **hardware-stored keys** are a better line of defense!
- Alice → Bob can use the same key as Bob → Alice
  - **False:** **never reuse keys**; each direction gets its own key!

Questions?

This time on CS 4440...

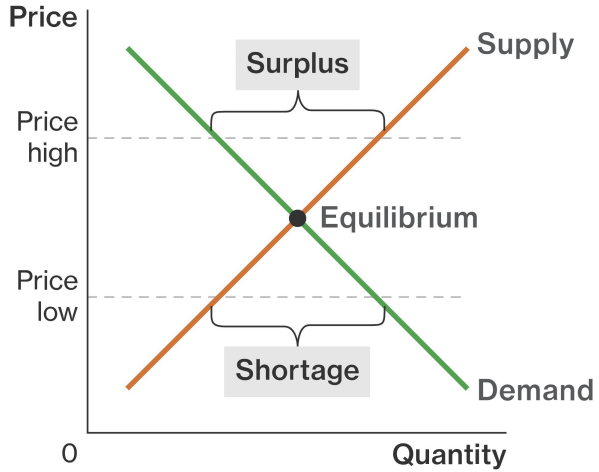
Cryptocurrency

Why does money have value?

Because others think it does!

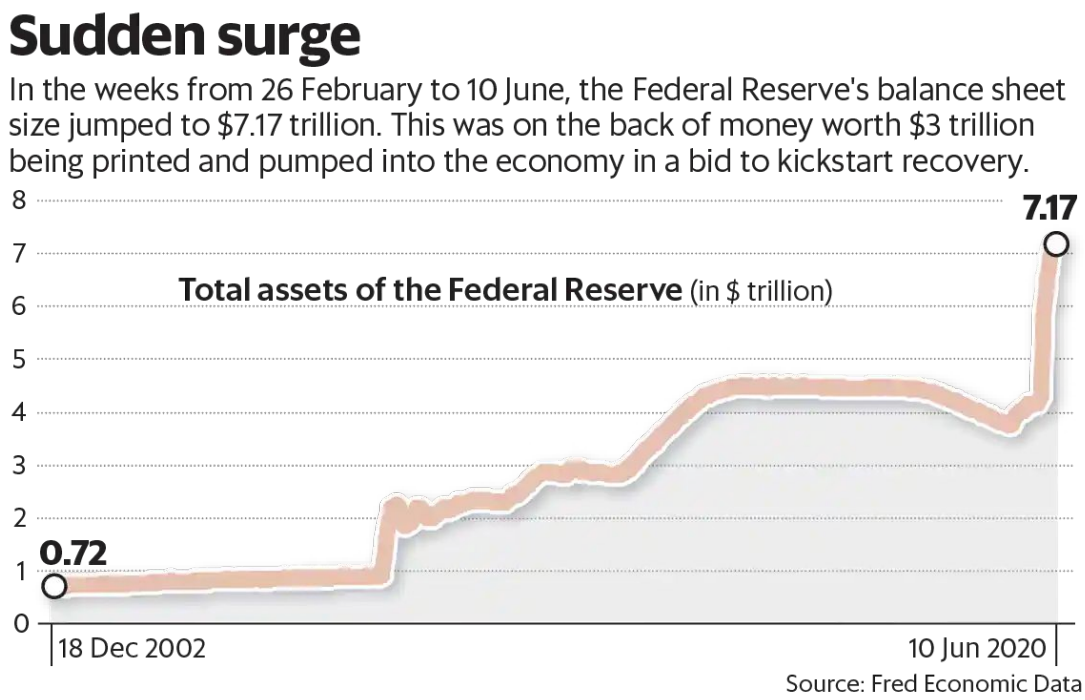
- **Demand:** belief in money's value; i.e., it can be exchanged for real things
- **Supply:** amount of money that actually exists

```
void government() {
    while(true)
        print money;
}
```



Inflation since Covid-19

- How does this affect your personal savings?



What if...

- **Challenge:** Create our own money that is not controlled by any single government and doesn't require huge start-up investment

- **Why?**
  - No need to waste resources printing or securing paper money
    - Save the trees!
      - Sort of...
  - Not controlled by a single entity (e.g., government)—**hopefully**
  - Manageable privacy
  - End-user can “print” their own money!

We are Satoshi Nakamoto!

- **Introducing... Cryptocurrency**
  - Invented in 2008 (Bitcoin) by **Satoshi Nakamoto**
  - His/their real identify remains a mystery
- **Key Principles**
  - Integrity
  - Distributed Consensus
  - **Cryptographic Hash Function**
  - Public-key Crypto
  - Proof-of-Work

DOGECOIN DOGE

Cryptocurrency Challenges

1. Keeping records without centralizing trust
2. Maintaining anonymity for all users
3. Preventing fake transactions
4. Preventing duplicate transactions
5. “Printing” new “money”

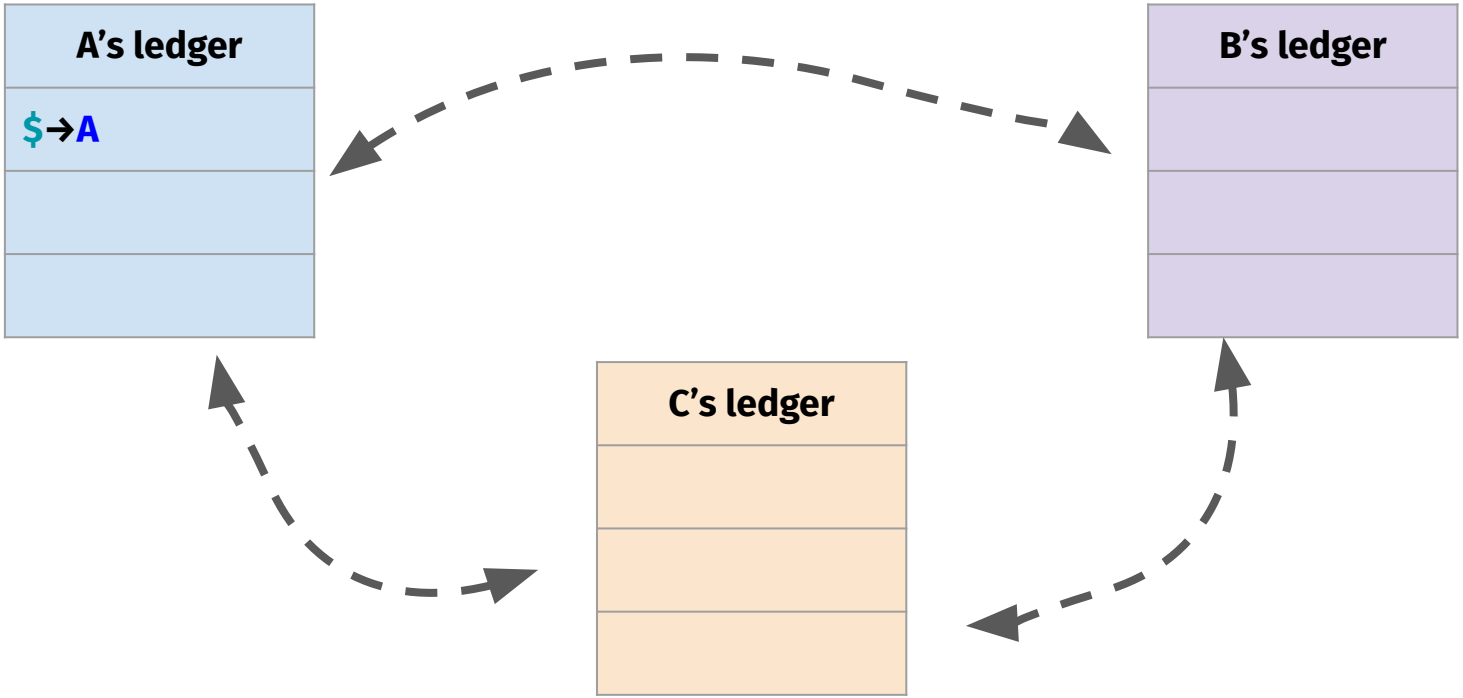
Traditional Banking

- Uses a **centralized ledger**
  - Cannot go to **Bank A** and withdraw all your money…
  - … then go to **Bank B** and withdraw it all over again!
  - **Fun fact:** originally on-paper
- Tracks customer accounts
  - Only have as much **$$$** as bank (and FDIC) say you do

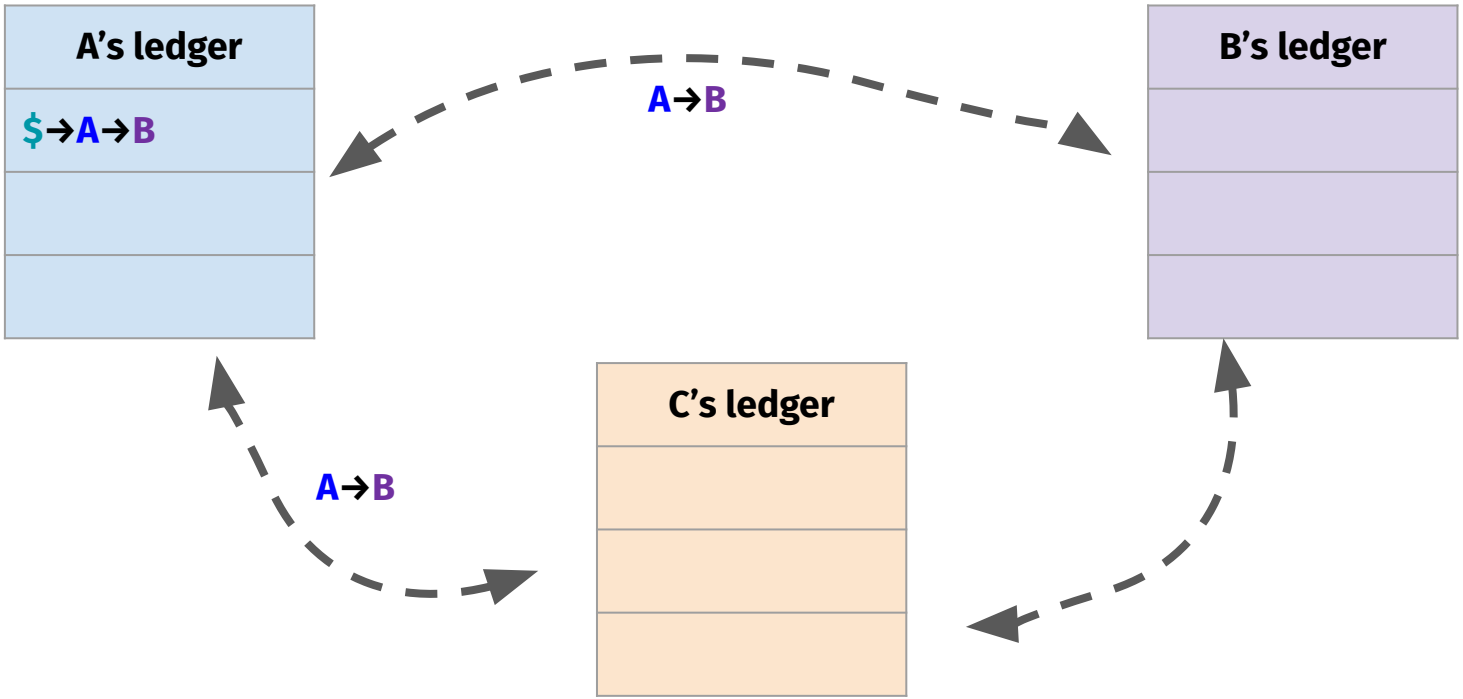
A Decentralized Ledger

- **Why?**
- **How to build it?**

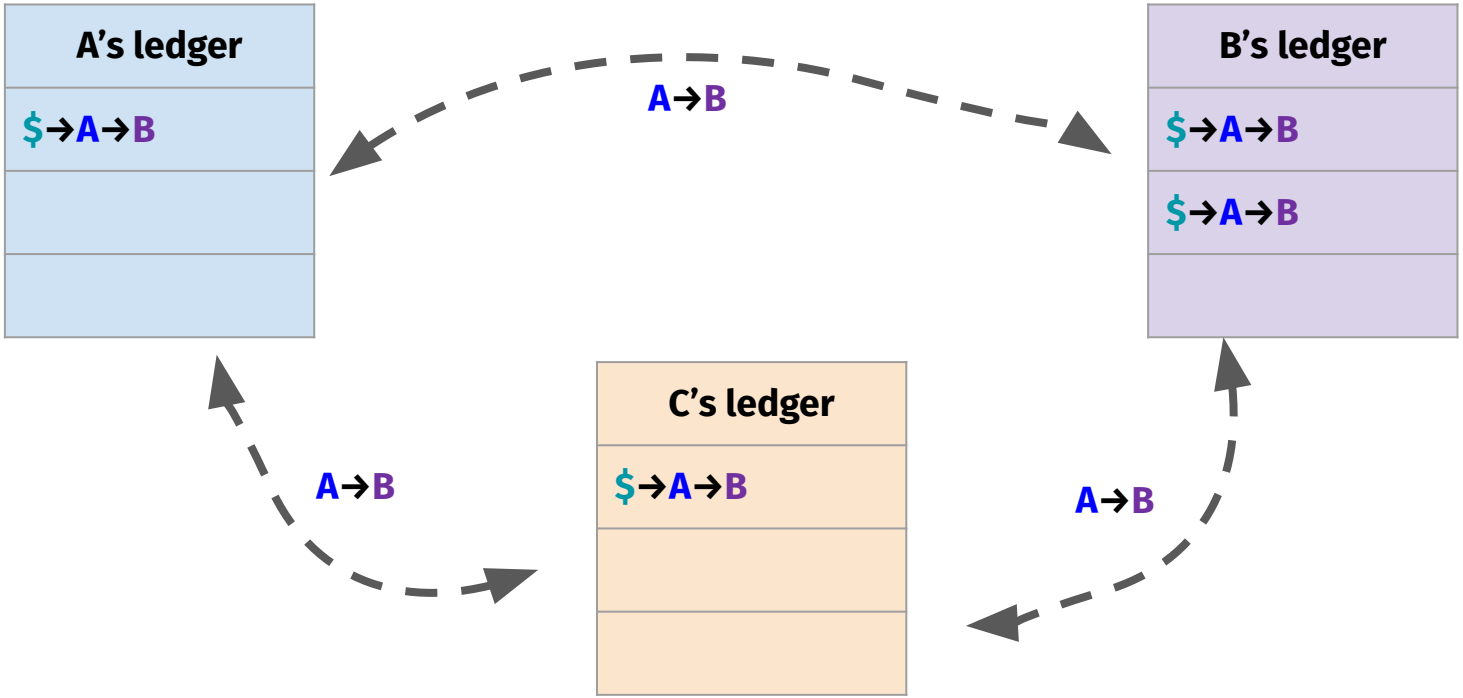
A has some $



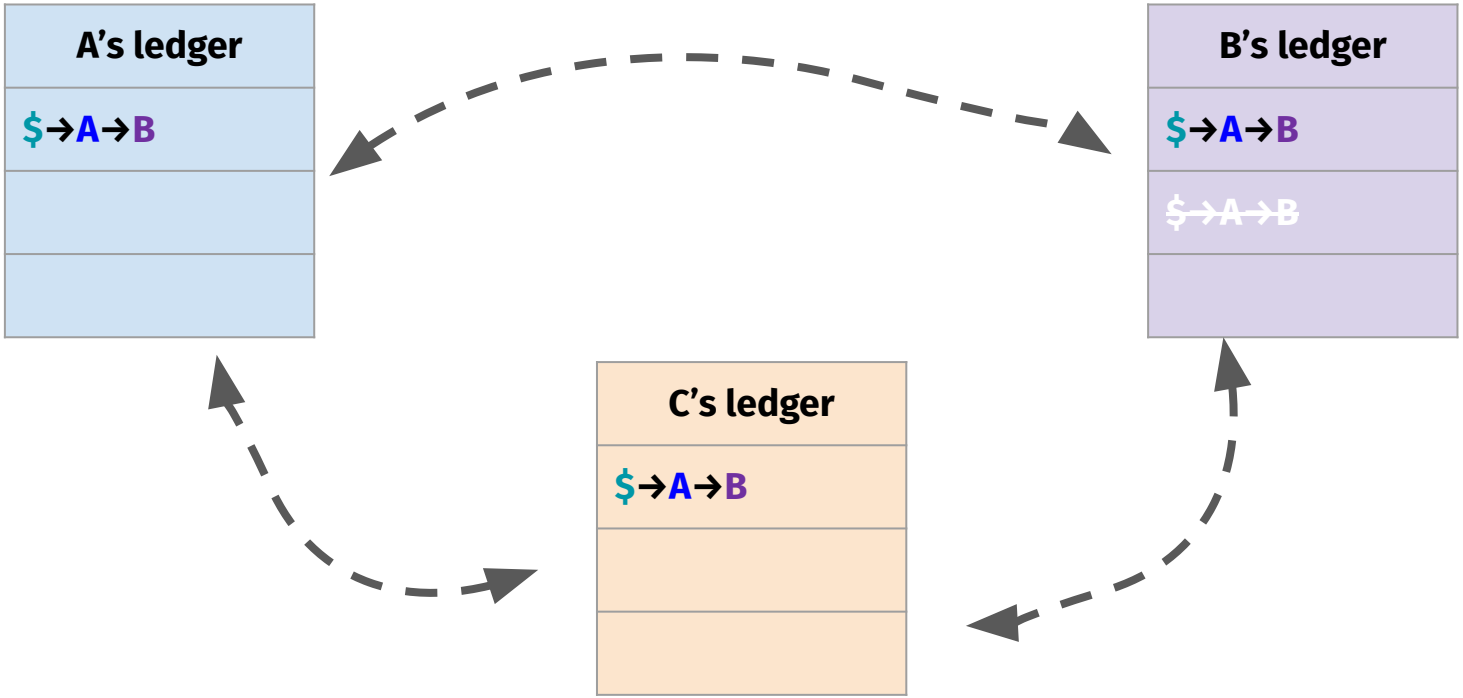
A exchanges $ for pizza from B



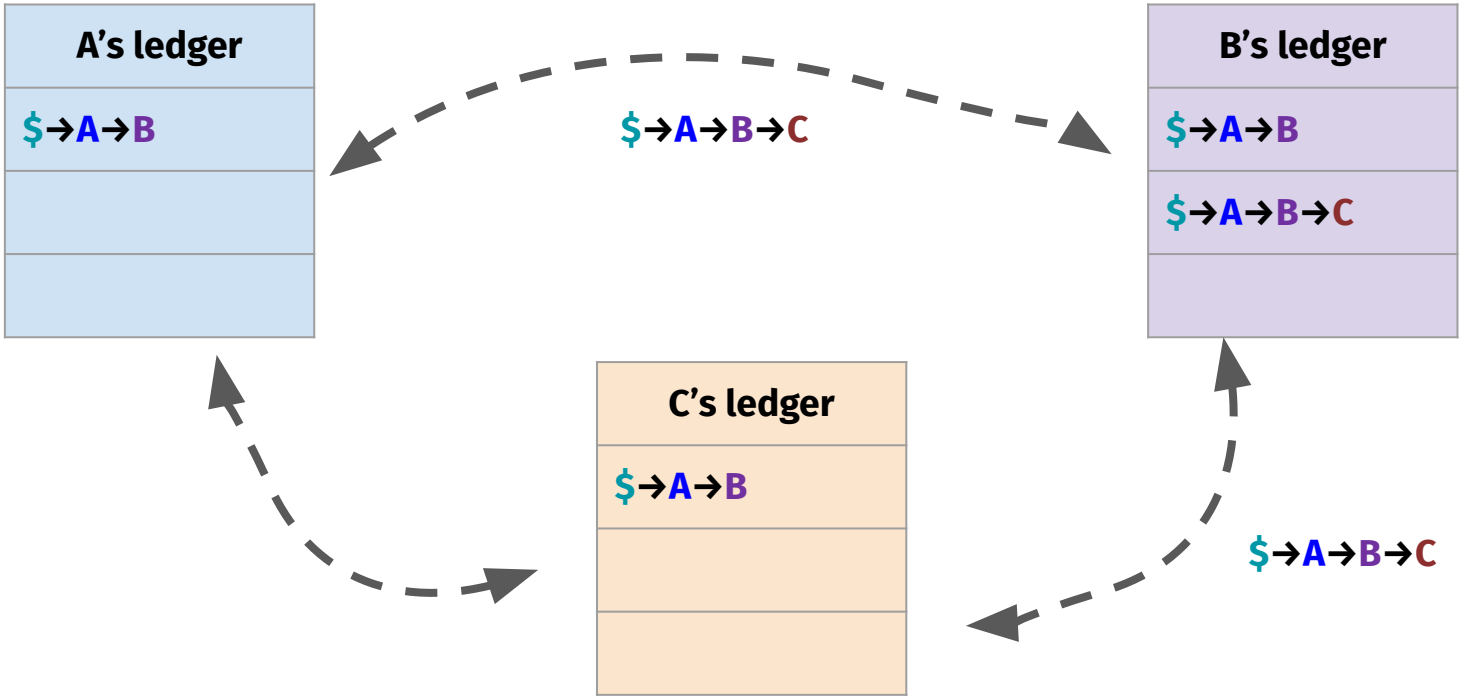
A exchanges $ for pizza from B



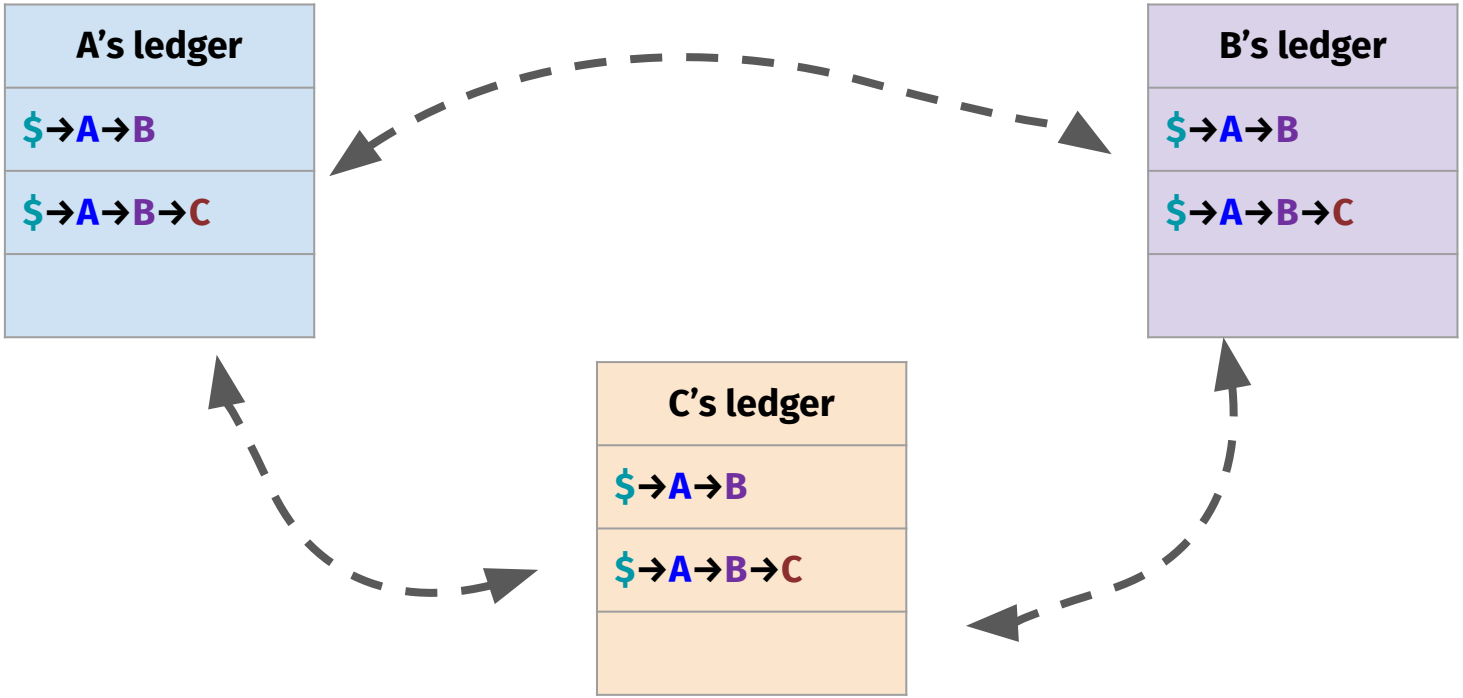
A exchanges $ for pizza from B



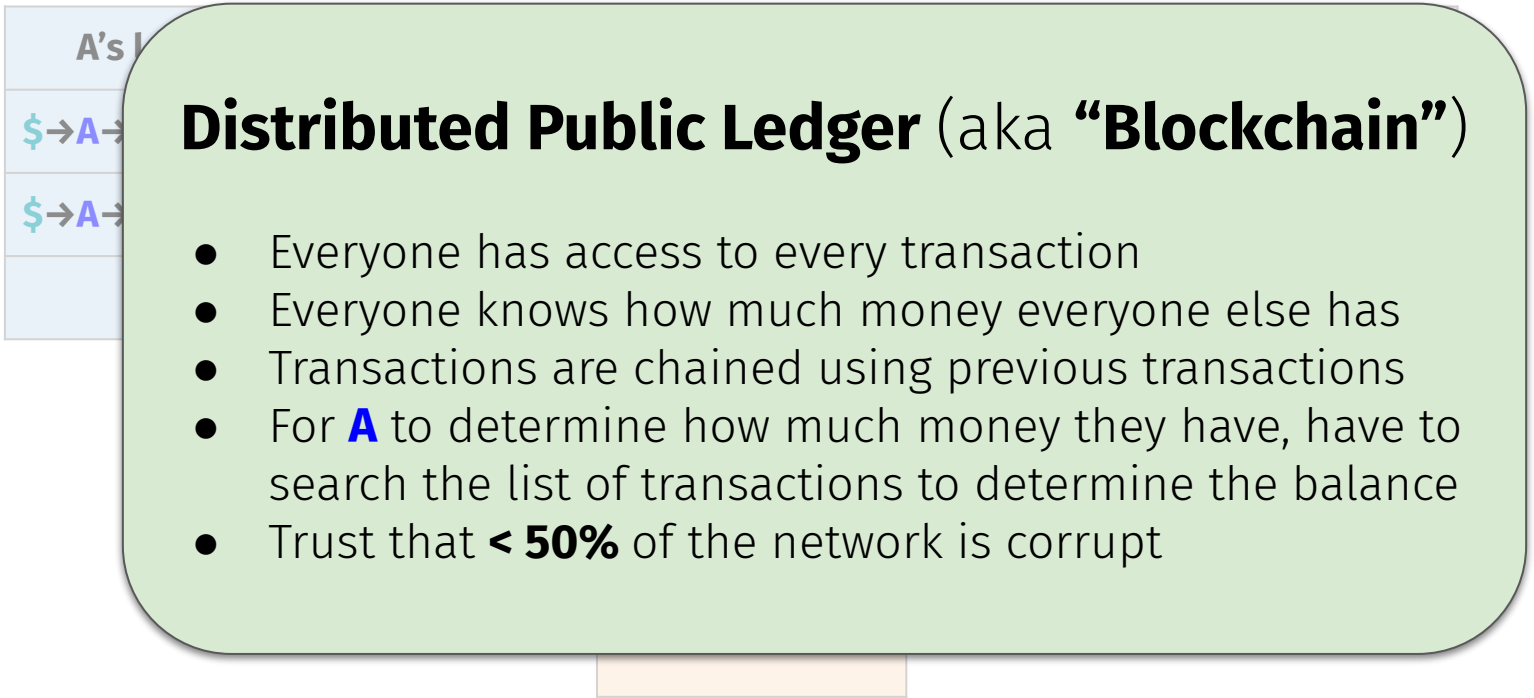
B exchanges $ for beer from C



B exchanges $ for beer from C



B exchanges $ for beer from C



Cryptocurrency Challenges

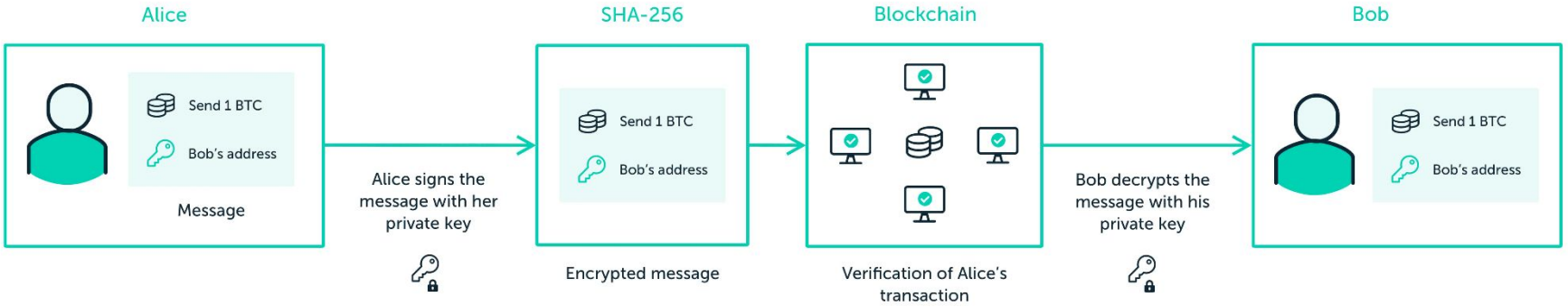
1. ~~Keeping records without centralizing trust~~
2. Maintaining anonymity for all users
3. Preventing fake transactions
4. Preventing duplicate transactions
5. “Printing” new “money”

Anonymity

- **Why?**
- **How to enforce it?**
  - Use what we've learned so far?

Transactions

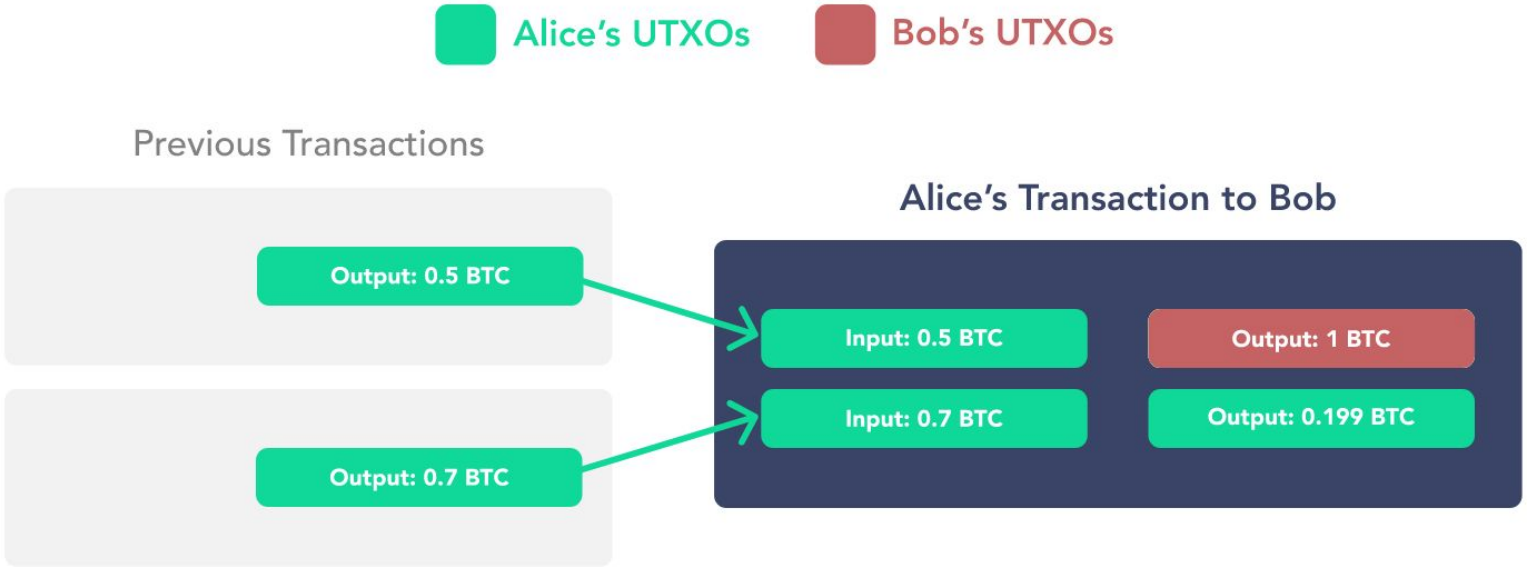
- **Key idea:** use **public-key crypto**
  - … instead of real identities
  - Derive shared secret key through “public” conversation



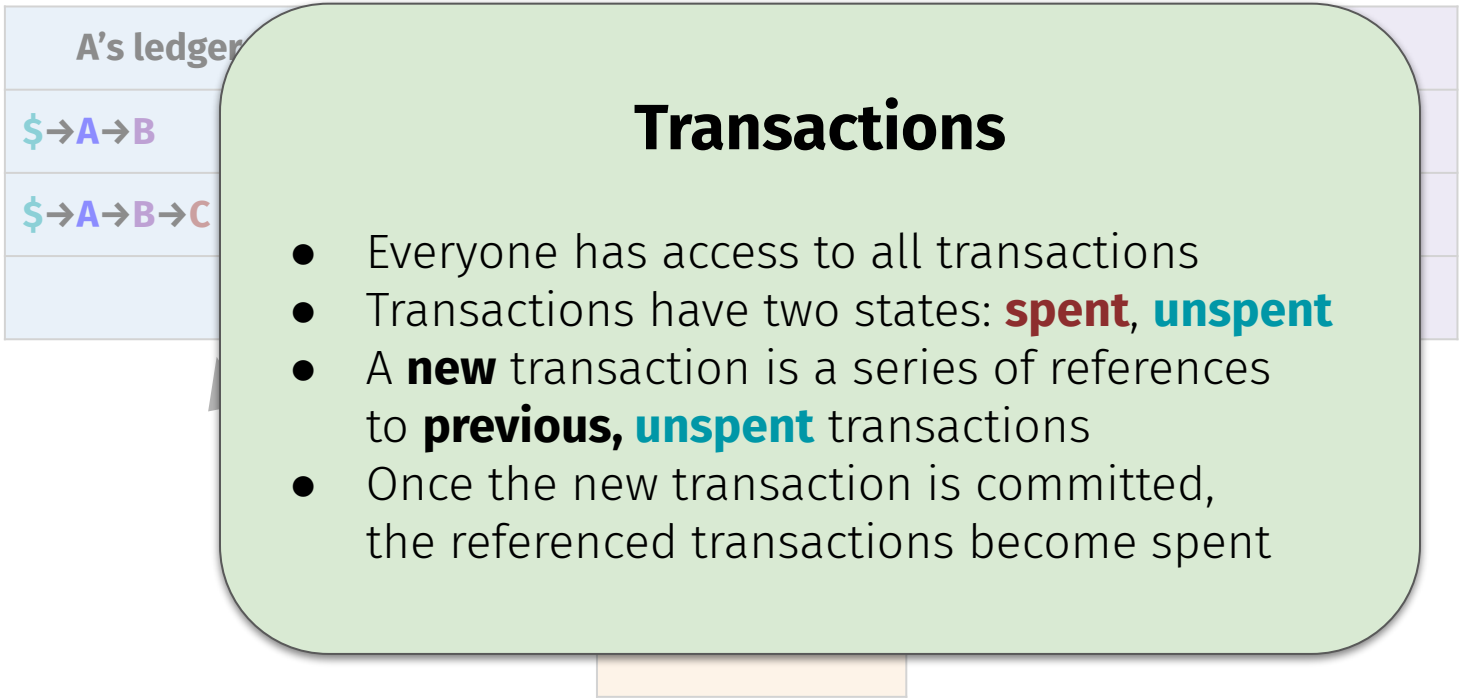
Transactions

- **What really is a transaction?**
  - Send money to everyone in a locked box
  - Only the intended has the key to open it
  - Everyone has access to that locked box **forever**
- If you **figure out** someone's private key
  - You can access money inside **any box** it opens
- If you **forget/lose** your private key
  - You lose access to **any box** that it would open

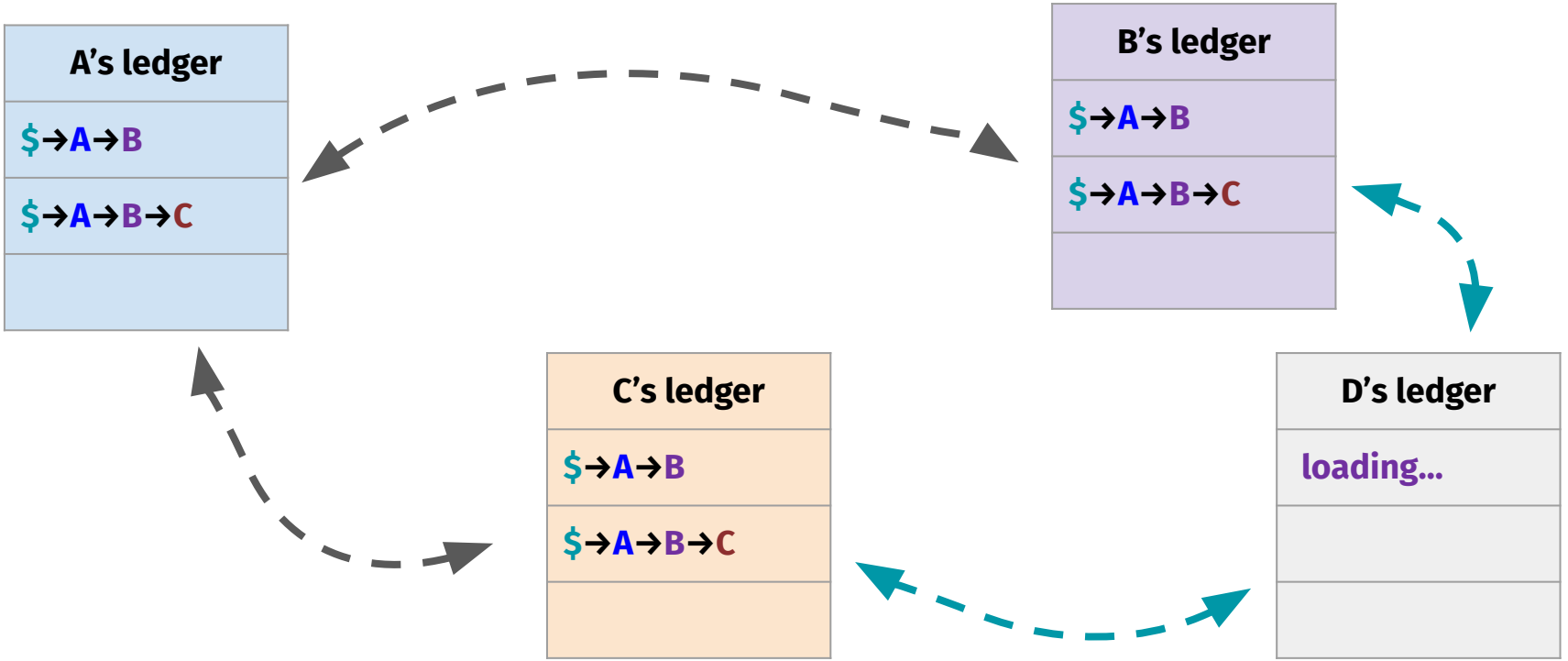
Unspent, Spent Transaction Model



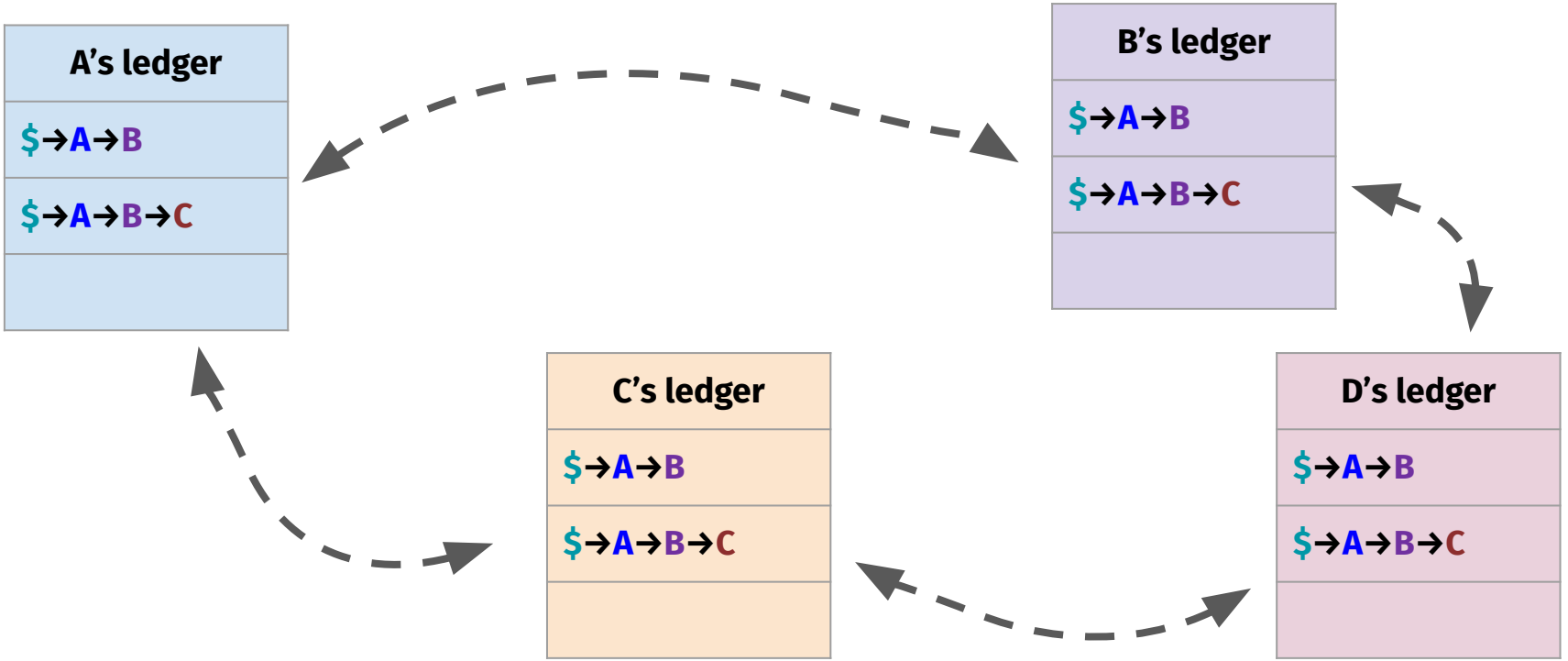
Transactions



New node? Download all transactions



New node? Download all transactions



New node? Download all transactions

Anonymous Transactions

- Nodes/clients can **come and go at will**
- There is no network authentication
- As long as you have the **private key** for the bitcoin sent to you, you can create a new transaction using that bitcoin
- Nothing prevents using a different priv/pub key pair for each incoming transaction
- **Thus, no need to use a unique identifier**

Cryptocurrency Challenges

1. ~~Keeping records without centralizing trust~~
2. ~~Maintaining anonymity for all users~~
3. Preventing fake transactions
4. Preventing duplicate transactions
5. “Printing” new “money”

Preventing Fake Transactions

- **Problem: malicious user** uses their ledger to create fake transactions where they are the recipient

Preventing Fake Transactions

- **Problem: malicious user** uses their ledger to create fake transactions where they are the recipient

- **Solution:** the **real** sender signs the transaction with their private key
  - **Unless key captured**, can't fool
  - Relying on mathematical hardness

Preventing Duplicate Transactions

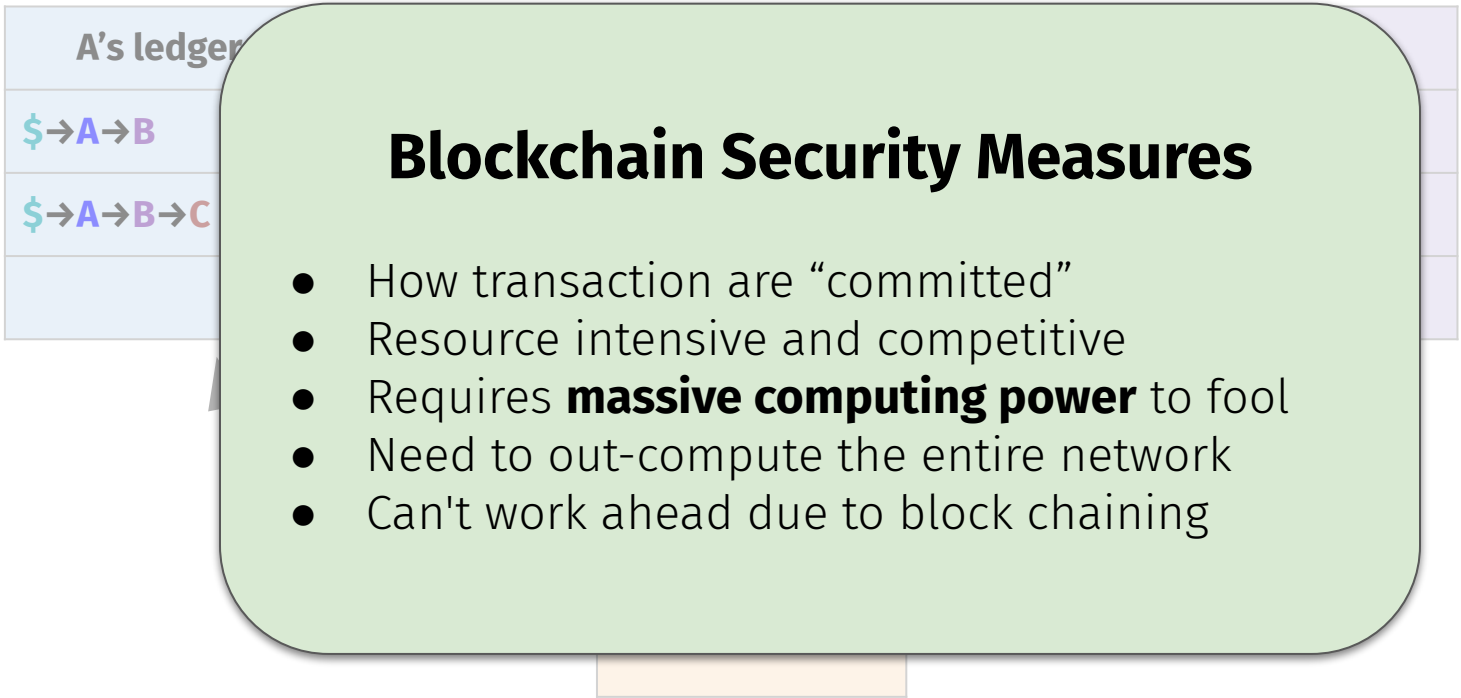
- **Problem: malicious user** creates a fake ledger, tries to convince rest of network that it is really legitimate

Preventing Duplicate Transactions

- **Problem: malicious user** creates a fake ledger, tries to convince rest of network that it is really legitimate

- **Solutions:**
  - Make it **expensive** and **competitive** to commit your version of the ledger to the entire network (**dist. consensus**)
  - In cases of mismatched ledgers, the **longer one wins**
  - Make future ledger commits depend on **past ledger commits**

The Blockchain



Cryptocurrency Challenges

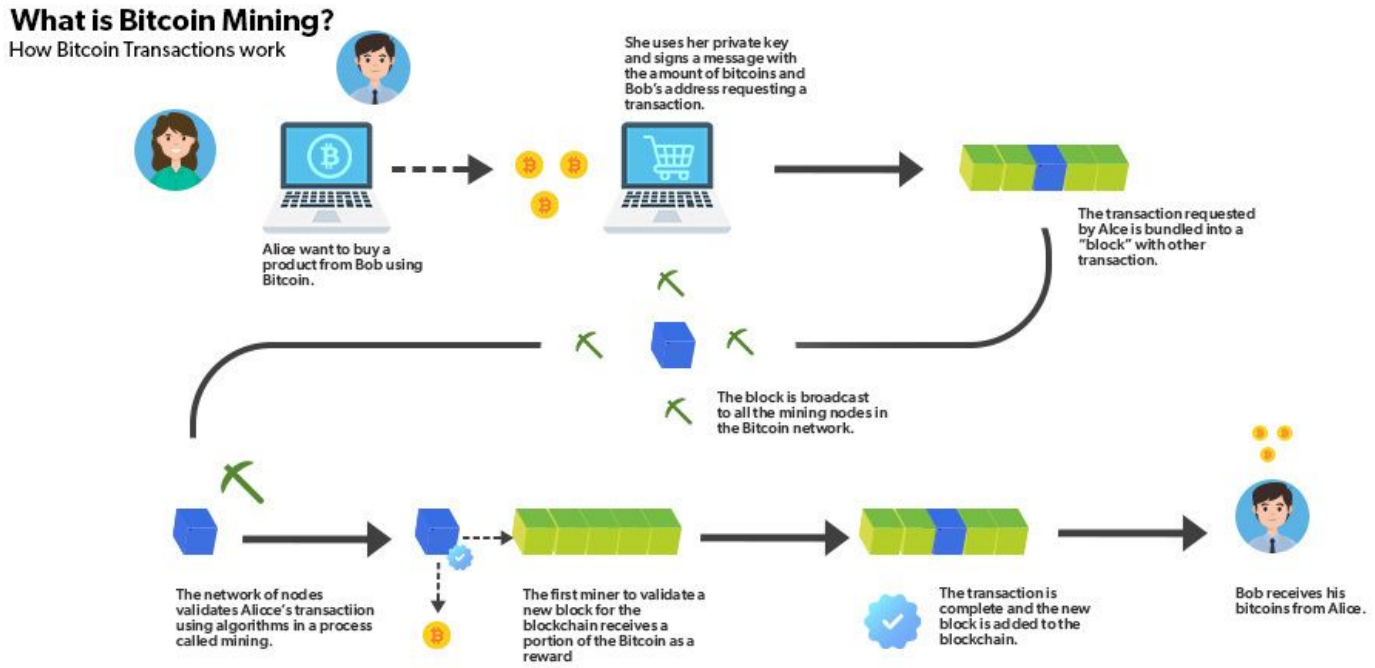
1. ~~Keeping records without centralizing trust~~
2. ~~Maintaining anonymity for all users~~
3. ~~Preventing fake transactions~~
4. ~~Preventing fake transactions~~
5. “Printing” new “money”

Creating new Money

- **Super high-level idea:** reward whoever "validates" a transaction
  - Validators are called "miners"
  - Given a small commision

- Meant to be a **fair process** that does not cost money
  - Anyone can start mining!

Creating new Money



Is cryptocurrency really fair?

Always!

0%

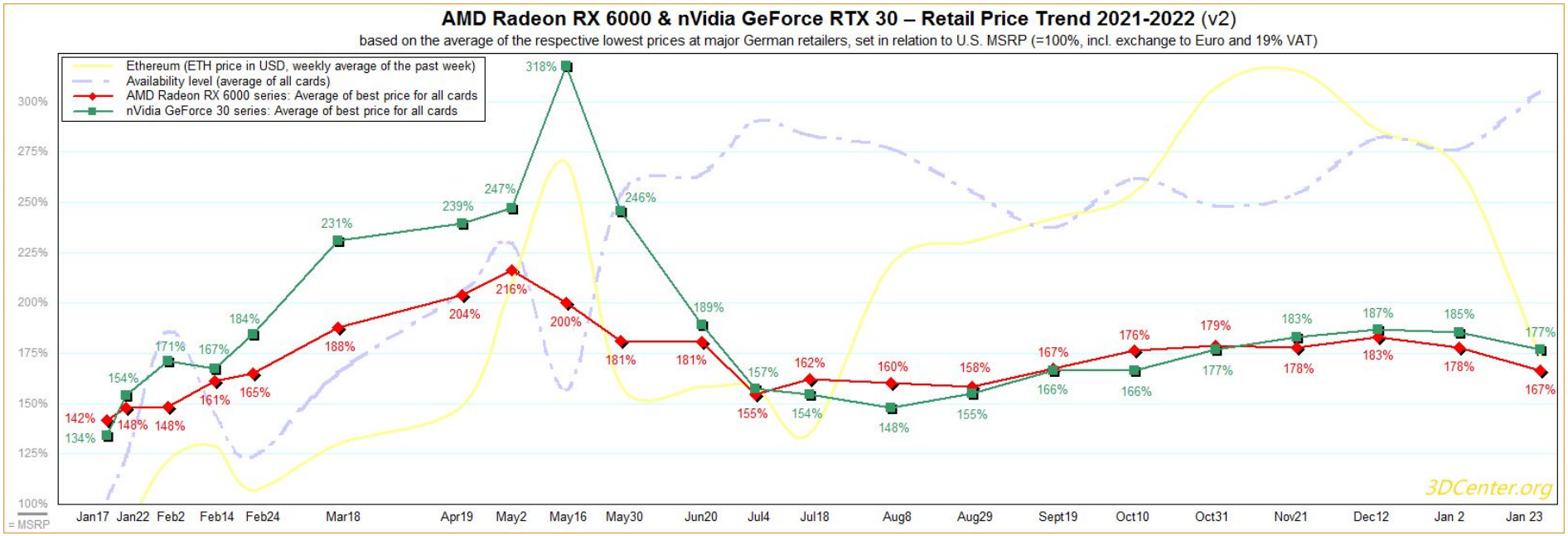
Maybe?

0%

Never!

0%

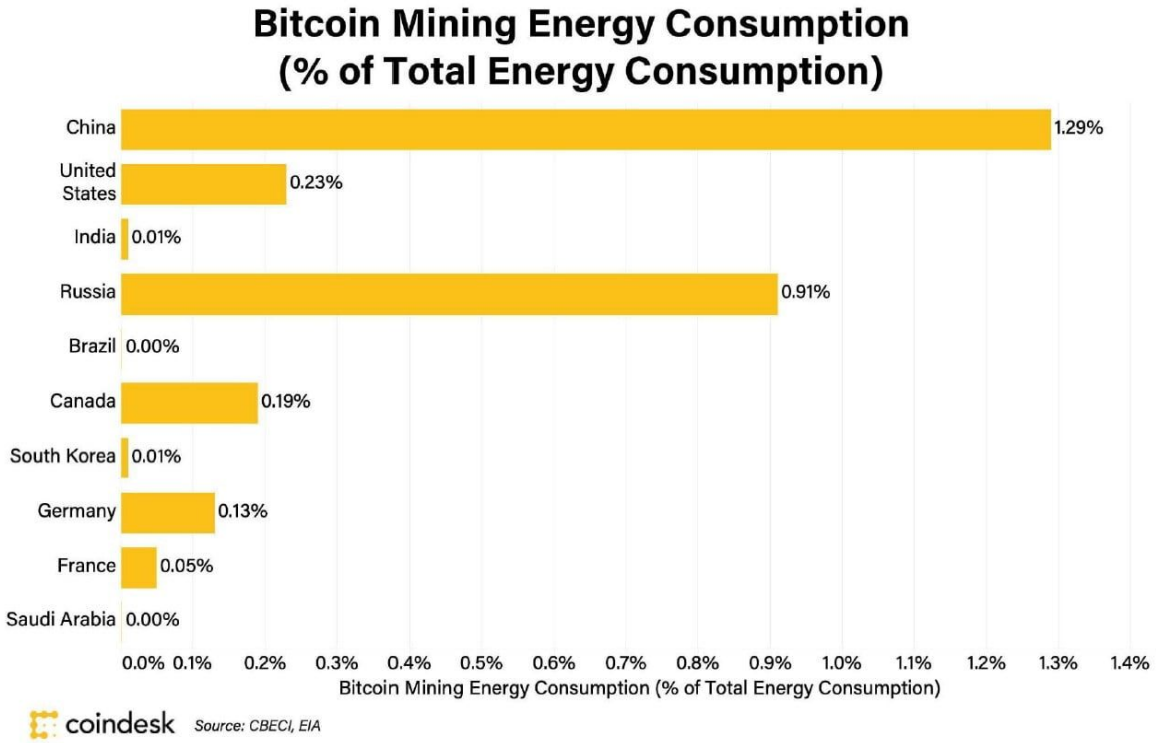
Is it really fair?



Is it really fair?

Is it really fair?

Is it really fair?



Is it really fair?

Bitcoin remains the most polluting cryptocurrency in 2022 while Ethereum massively reduced emissions: report

- Bitcoin accounted for 86.3 million tons of CO2 emissions in 2022, according to Forex Suggest report
- Cryptocurrency Litecoin replaced Ethereum as the second-most polluting token on Forex Suggest's ranking

The cryptocurrency industry is facing increasing regulatory and investor scrutiny as it matures, and strong reporting and assurance practices are defining resilient companies. One area that is top of mind for stakeholders is the crypto industry's impact on the environment, particularly as focus on environmental, social, and governance (ESG) strategies and reporting increases. Crypto mining is an energy-intensive activity, and its environmental impact is prompting investor questions about energy sourcing.

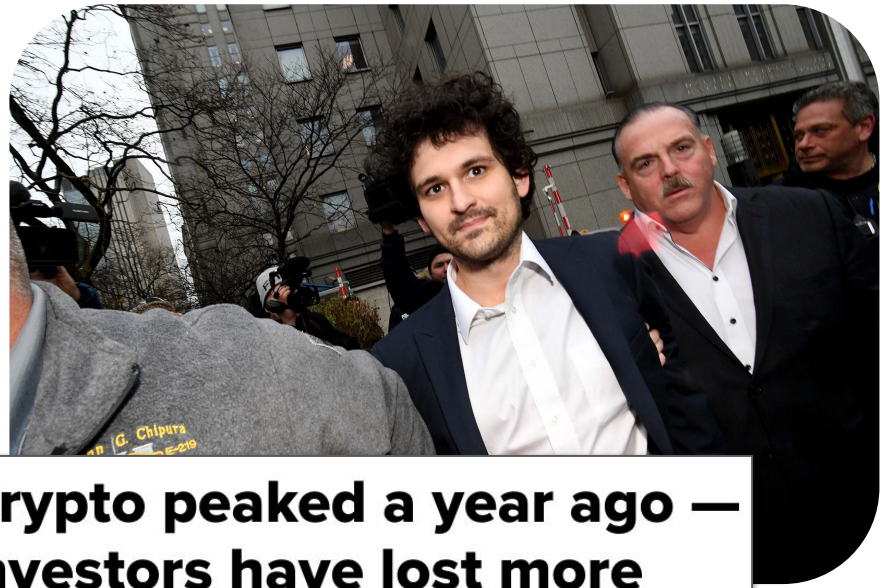
Mining

- In practice, not really fair...
  - Hardware and GPU cost
  - Electricity cost
  - Environmental cost
  - More money gives an advantage!

Mining

- In practice, not really fair…
  - Hardware and GPU cost
  - Electricity cost
  - Environmental cost
  - More money gives an advantage!

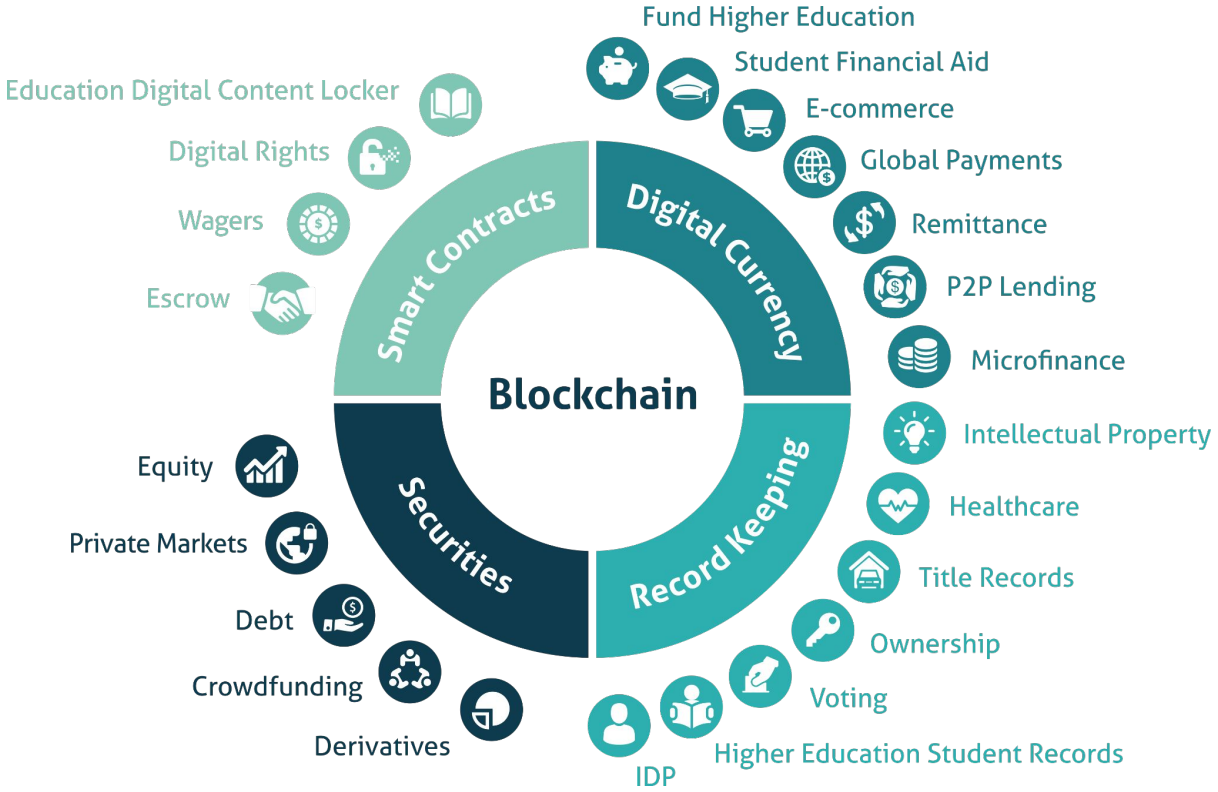
- **Don’t buy into the hype!**
  - Stock market has more certainty
    - Not an official endorsement!



Other Blockchain Applications

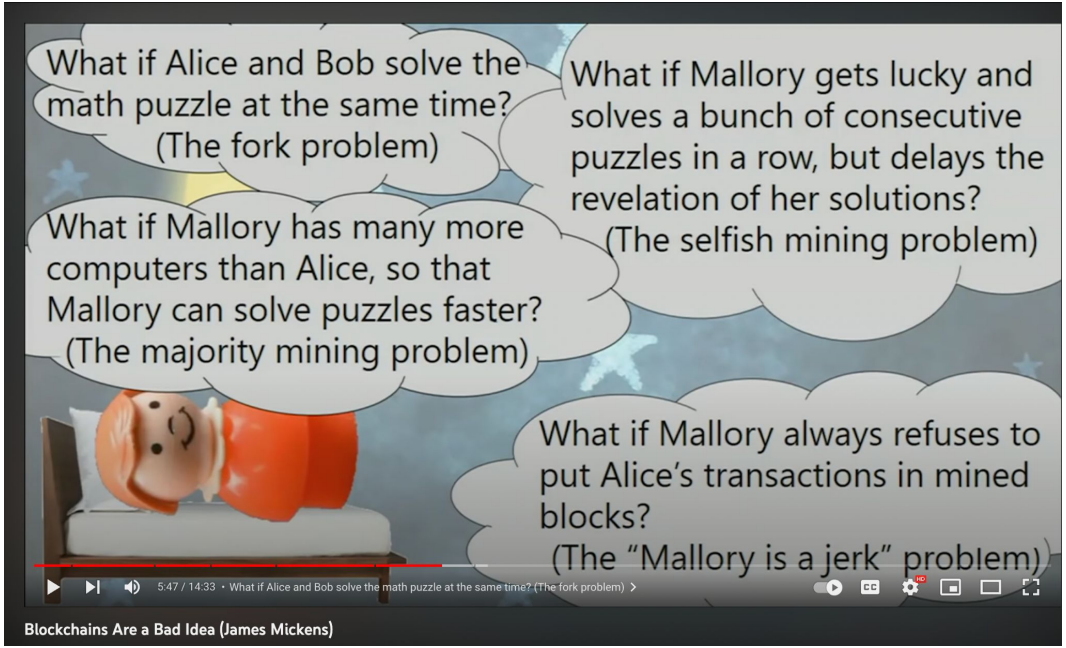
- **Examples?**

Other Blockchain Applications



Not everything needs to be Blockchained!

https://www.youtube.com/watch?v=mDwUJa4_IJE



Next time on CS 4440...

Intro to Application Security