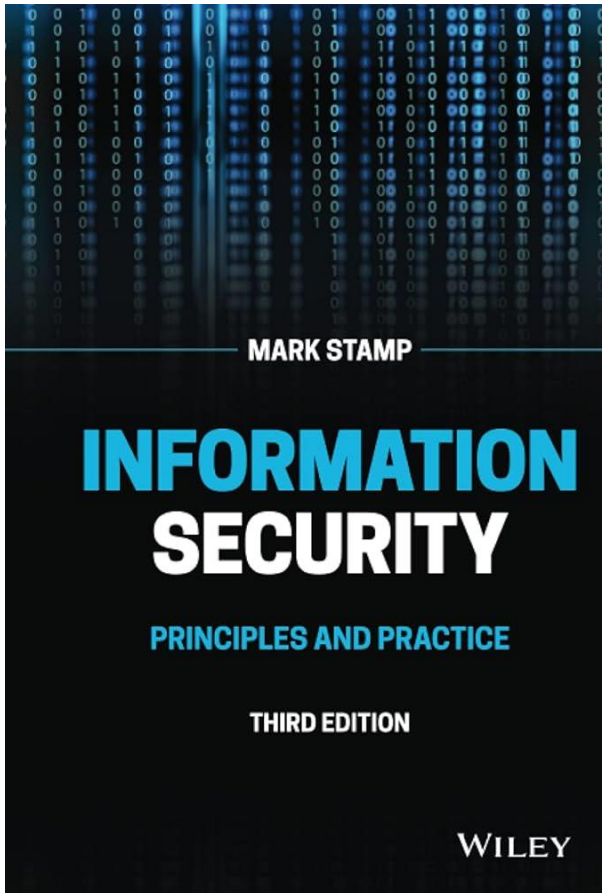
بسم الله الرحمن الرحیم

# مبانی رایانش امن

## جلسه ۱۲

مجتبی خلیلی
دانشکده برق و کامپیوتر
دانشگاه صنعتی اصفهان


MARK STAMP
INFORMATION SECURITY
PRINCIPLES AND PRACTICE
THIRD EDITION
WILEY

- فصل ؟ استمپ

# Blockchain

# Digital Cash is not New

- DigiCash Inc., founded in 1989
  - Based on Chaum's "blind signatures"
  - Strong crypto that ensures anonymity
- Back then, the "killer app" was thought to be ***micropayments***
  - Users on the Internet pay only a tiny amount (fraction of cent) for something
- DigiCash declared bankruptcy in 1998

# Digital Currency

- We want create an all-digital currency
  - Like $ or ¥ or € or …., but "better"
- Real cash is (relatively) anonymous
  - So digital currency should be too
- Digital currency is "better" since…
  - No central authority (i.e., banks)
  - No government to issue currency, etc.

# Preliminaries: Ledgers

- ***Ledger*** is a book of financial accounts
- Suppose Alice, Bob, Charlie, Trudy play weekly poker game online
- They all insert ledger entries such as,

  "Bob owes Alice $10", "Charlie owes Trudy $30", "Trudy owes Alice $25", and so on
- Once a month, they meet and settle up
- Any possible problems here?

# Signed Ledger Entries

- How to prevent Trudy from inserting, say, "Bob owes Trudy $1M" ?
- So, let's require **digital *signatures***
  - For ledger entry to be valid, Bob must sign "Bob owes Alice $10", Trudy must sign "Trudy owes Alice $25", and so on …
- Then we know ledger entries are valid
  - That is, the payer agrees to pay

# Signed Ledger

- Ledger now looks like
  - [Bob owes Alice \$10]$_{Bob}$
  - [Charlie owes Trudy \$30]$_{Charlie}$
  - [Trudy owes Alice \$25]$_{Trudy}$
  - and so on …
- And we know ledger entries are valid
- But, still some problems here…

# Signed Ledger in Detail

- As an aside, note that signatures on previous slide really look like
  - $(M_1,[h(M_1)]_{Bob})$, where $M_1$="Bob owes Alice $10"
  - $(M_2,[h(M_2)]_{Charlie})$, $M_2$="Charlie owes Trudy $30"
  - $(M_3,[h(M_3)]_{Trudy})$, $M_3$="Trudy owes Alice $25"
  - And so on …
- We'll use the shorthand on previous slide

# Ledger Duplication

- Still, nothing to prevent Trudy from duplicating a line…
  - $[\text{Bob owes Alice \$10}]_{\text{Bob}}$
  - $[\text{Charlie owes Trudy \$30}]_{\text{Charlie}}$
  - $[\text{Trudy owes Alice \$25}]_{\text{Trudy}}$
  - $[\text{Charlie owes Trudy \$30}]_{\text{Charlie}}$
- Signatures are still all valid
- How to prevent this attack?

# Unique Ledger Entries

- Include unique transaction numbers
  - [1, Bob owes Alice $10]$_{Bob}$
  - [2, Charlie owes Trudy $30]$_{Charlie}$
  - [3, Trudy owes Alice $25]$_{Trudy}$
  - And so on…
- Why does this help?
- We will never have an exact duplicate
  - So any duplicate is invalid

# Ledger Prepayment

- How to be sure participants pay up?
- Can start with Alice, Bob, Charlie, and Trudy all putting money into the pot
- And don't allow any transaction that would result in negative balance
- Transaction must still be signed and …
- … now, nobody can "overdraw" account

# Ledger Prepayment Example


- Ledger example…
  - Alice has $100 // Alice's initial stake
  - Bob has $100 // Bob's intial stake
  - Charlie has $100 // Charlie's initial stake
  - Trudy has $100 // Trudy's initial stake
  - $[1, \text{Bob owes Alice \$10}]_{\text{Bob}}$ // **valid**
  - $[2, \text{Charlie owes Trudy \$30}]_{\text{Charlie}}$ // **valid**
  - $[3, \text{Trudy owes Alice \$25}]_{\text{Trudy}}$ // **valid**
  - $[4, \text{Trudy owes Bob \$120}]_{\text{Trudy}}$ // **invalid**

# Ledger Prepayment

- Note that we must know the ***entire*** transaction history
  - So that we can know current balances
  - Then we can be sure a given transaction does not cause user to be overdrawn
- This seems like kind of a hassle, but some big benefits come from it
  - As we will soon see…

# Eternal Ledger?

- Alice, Bob, Charlie, and Trudy could continue to settle accounts each month
- But, as the ledger currently stands, settling accounts is not necessary!
- We know the current balances, and no risk of anyone being "overdrawn"
- So, could play poker for months, years, or forever, without settling accounts

# Ledger as Currency

- This ledger can act as its own currency!
  - Need a cool symbol, let's use " § "
- Transactions ***within*** ledger are all in terms of the § "currency"
- Anyone can exchange ledger currency (i.e., § ) for $ or ¥ or € or …
  - But, such exchanges occur ***outside*** the ledger currency protocol

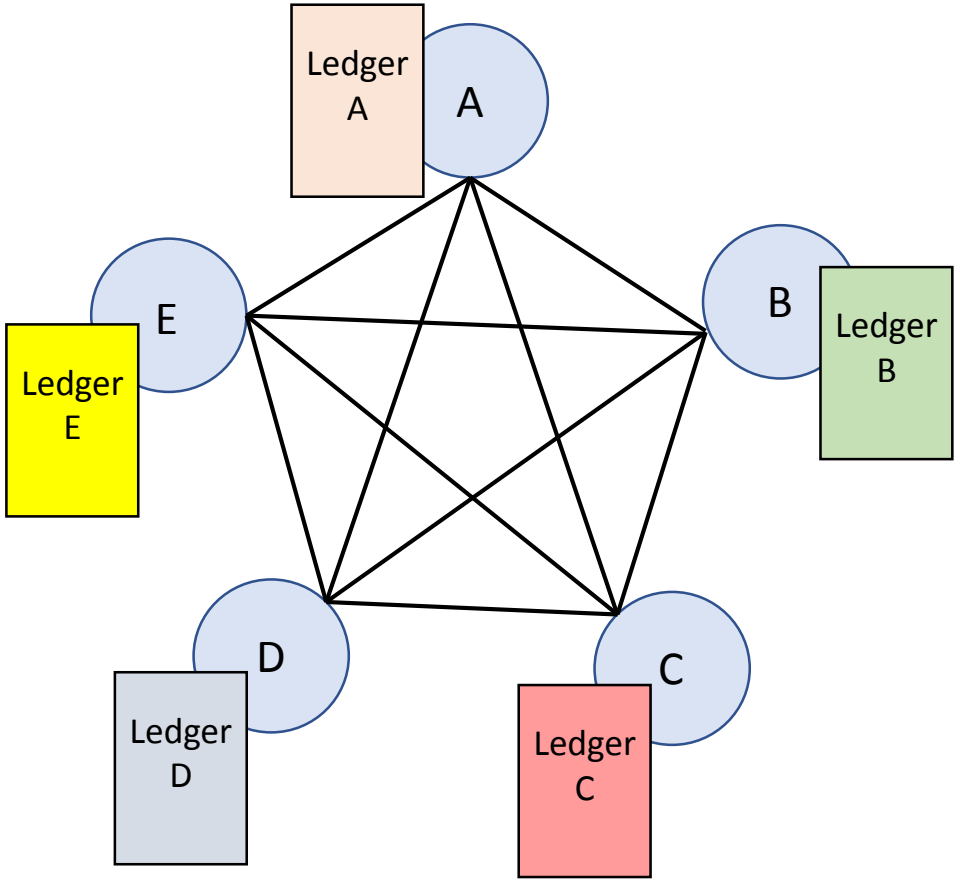
# Ledger Currency

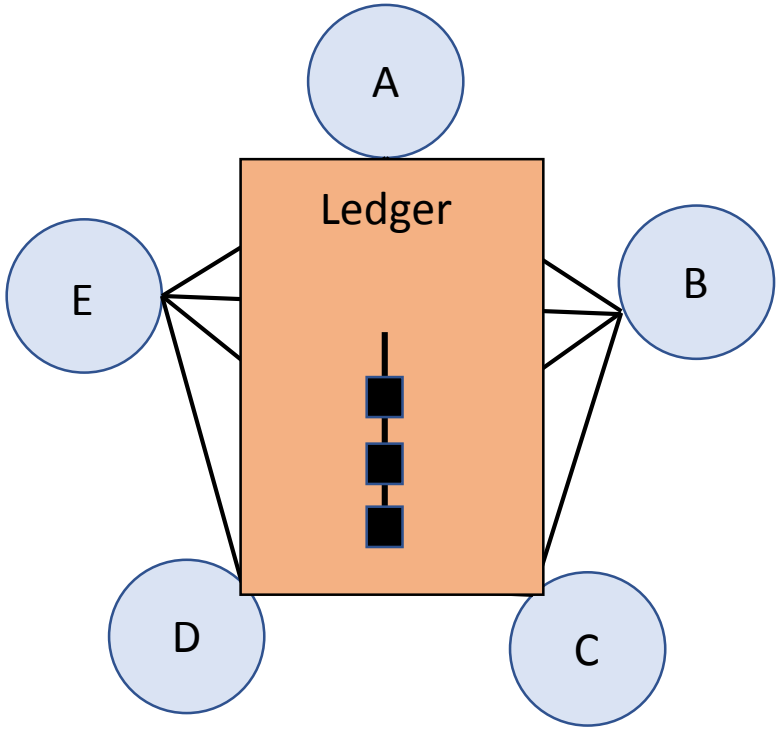
- For example, Alice could pay Bob $10 in real world dollars for, say, § 5 of currency in the ledger system
- Comparable to exchanging, say, $ for ¥
- The ledger is a history of transactions within the ledger currency system
- In fact, *the ledger is the currency*
  - This is the key insight for cryptocurrency

# Distributed Ledger



- The ledger is the currency
  - So who is in charge of the ledger?
  - A govt? The UN? A bank? An individual?


Ledger
A
A
B
Ledger
B
E
Ledger
E
D
Ledger
D
C
Ledger
C


A
Ledger
E
B
D
C

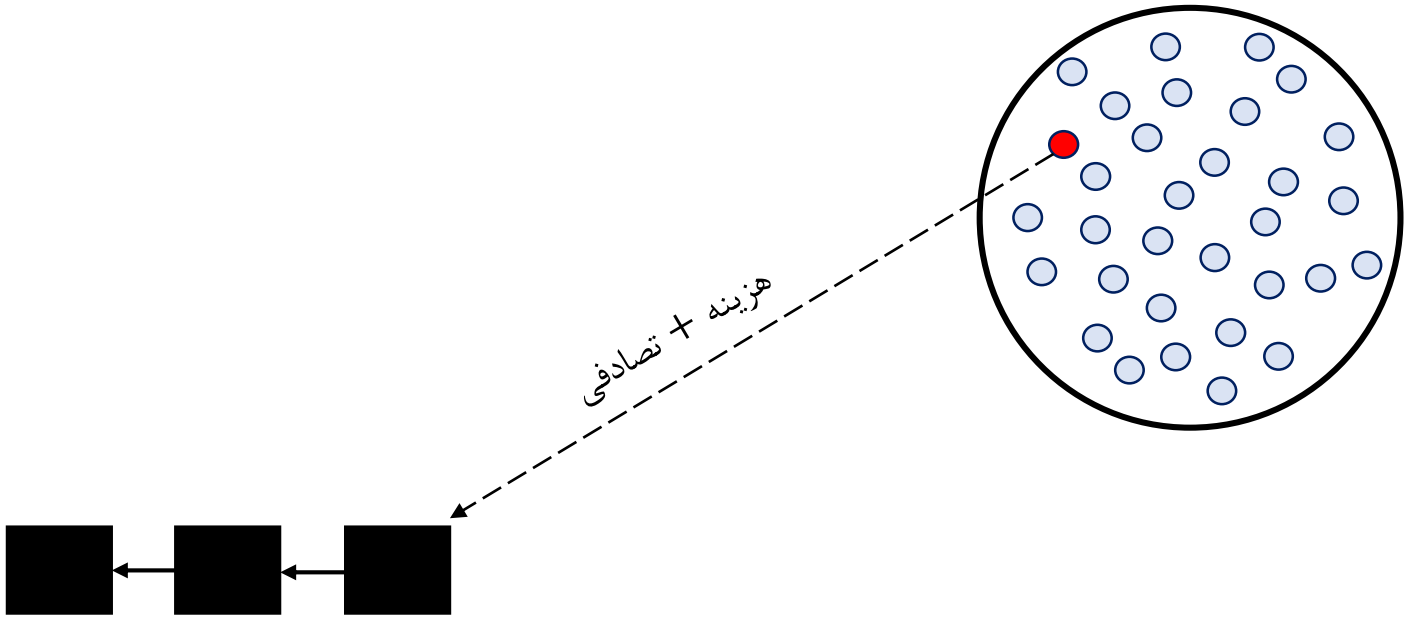
# Distributed Ledger



- The ledger is the currency
  - So who is in charge of the ledger?
  - A govt? The UN? A bank? An individual?
- We don't trust them, so let's put ***everybody*** in charge of the ledger
  - Anybody can have copy of ledger, anyone can add entries (there is a protocol…)
  - Protocol without a central authority!
- What problem(s) do you foresee?

# Distributed Ledger

1. Transactions must be signed
2. Nobody can be overdrawn
3. Transactions broadcast to everybody
   - How to have a consistent view of this distributed ledger?
   - Multiple ledgers can exist at any time
   - This is the heart of the issue for a distributed cryptocurrency (e.g. Bitcoin)


هزینه + تصادفی

# Preliminaries: Work

- How to measure (digital) ***work*** ?
- Our unit of work will be 1 hash
- Suppose that we have a hash function h(x) that generates an N-bit output
- Then randomly chosen input generates one of $2^N$ equally likely outputs
  - For any input R, have, $0 \leq h(R) < 2^N$
  - Different R yield uncorrelated hashes