Crytographic hash functions

How Bitcoin works?

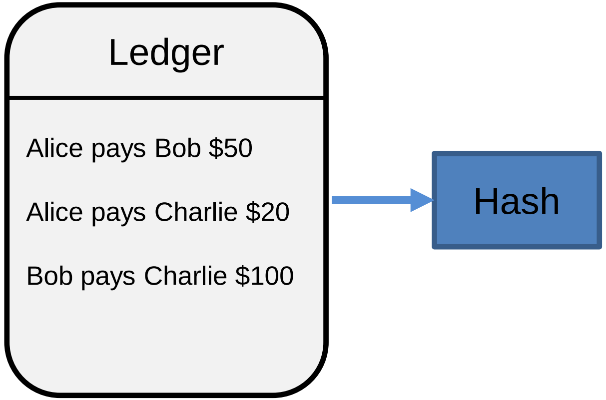
What is a hash function?

- A function (duh)
- With some properties (bucketing)

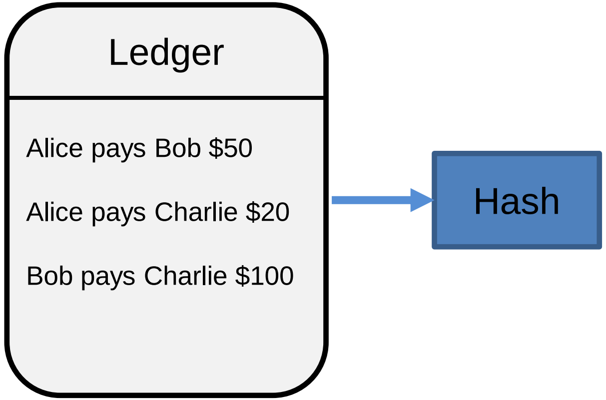
What is a hash function?

function/algorithm

What is a hash function?



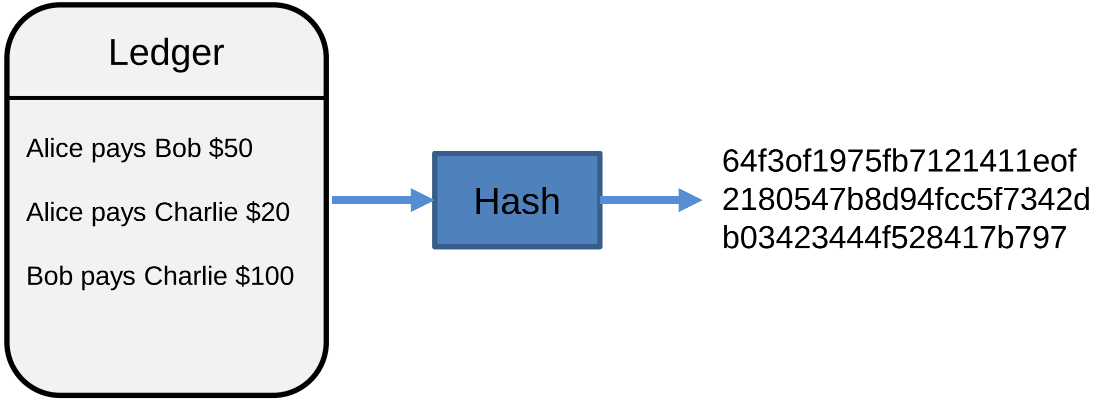
What is a hash function?



Arbitrary size

200GB

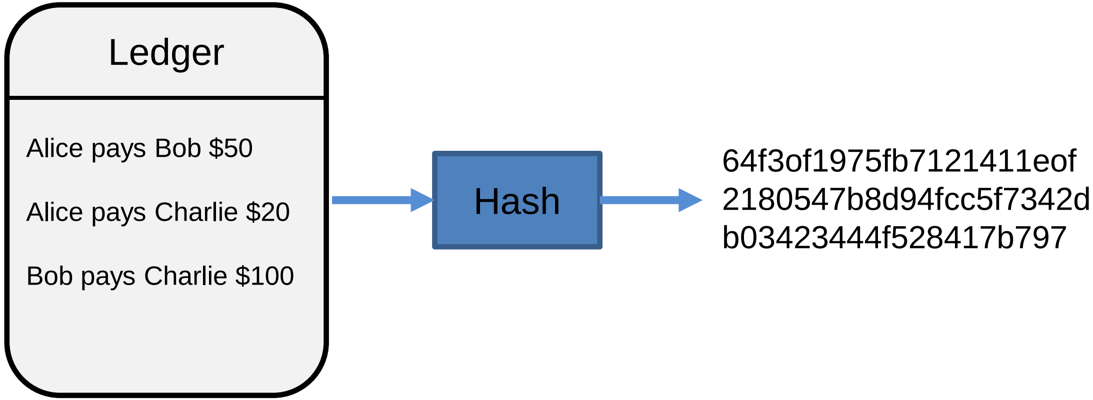
What is a hash function?



Arbitrary size

200GB

What is a hash function?



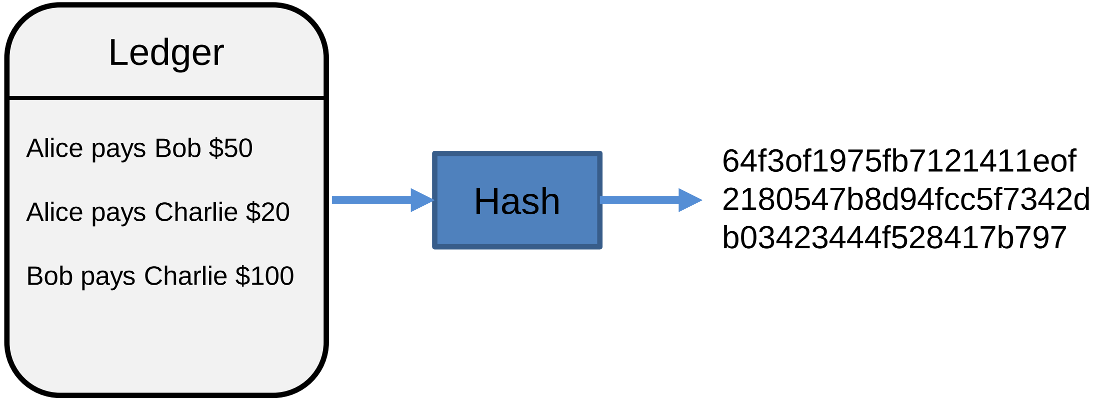
Arbitrary size

200GB

256 bits

What is a hash function?

Computable efficiently O(n)



Arbitrary size

200GB

256 bits

What is a hash function?

Function with these properties:

- Takes an arbitrary size input
- Produces a fixed size output
- Computes the output in O(n) time

What is a cryptographic hash function?

Hash function with some security properties:

- **Collision resistance**
- Hiding (hides the input)
- Puzzle friendliness (for mining)

Property 1

Collision resistance

A hash function H is **resistant to collissions** if:

- It is not *feasible to find* two inputs *x,y*
- $x \neq y$
- $H(x) = H(y)$

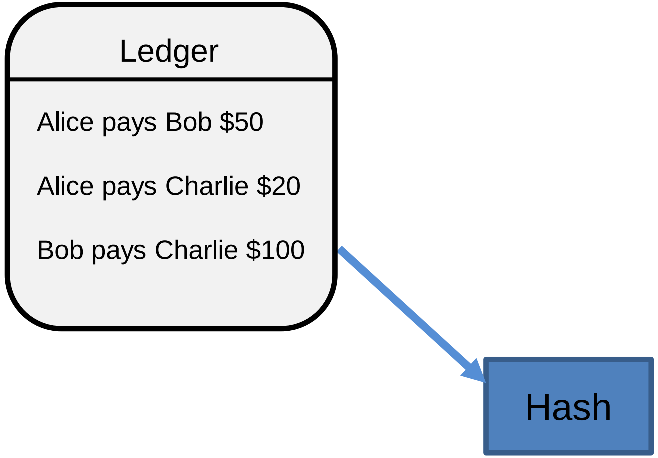
Property 1

Collision resistance

Ledger
Alice pays Bob $50
Alice pays Charlie $20
Bob pays Charlie $100

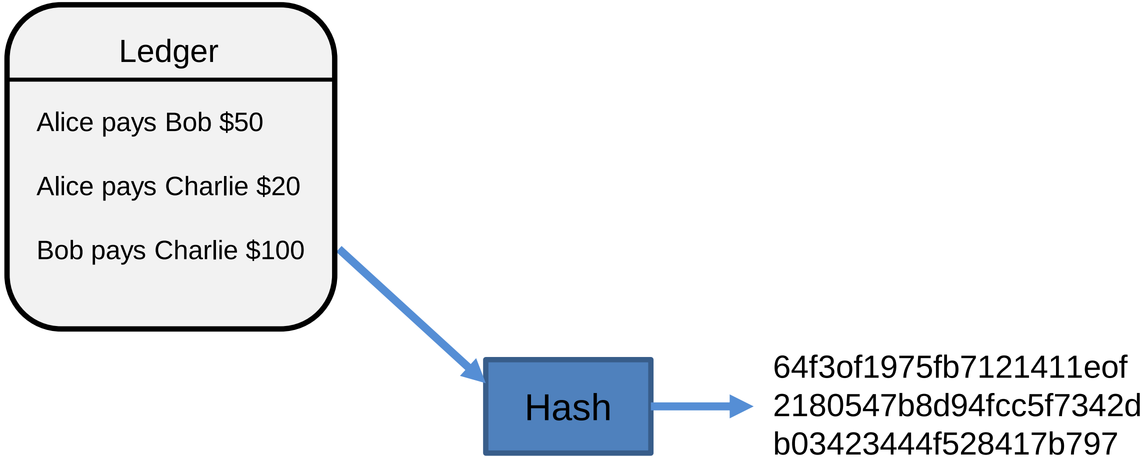
Property 1

Collision resistance



Property 1

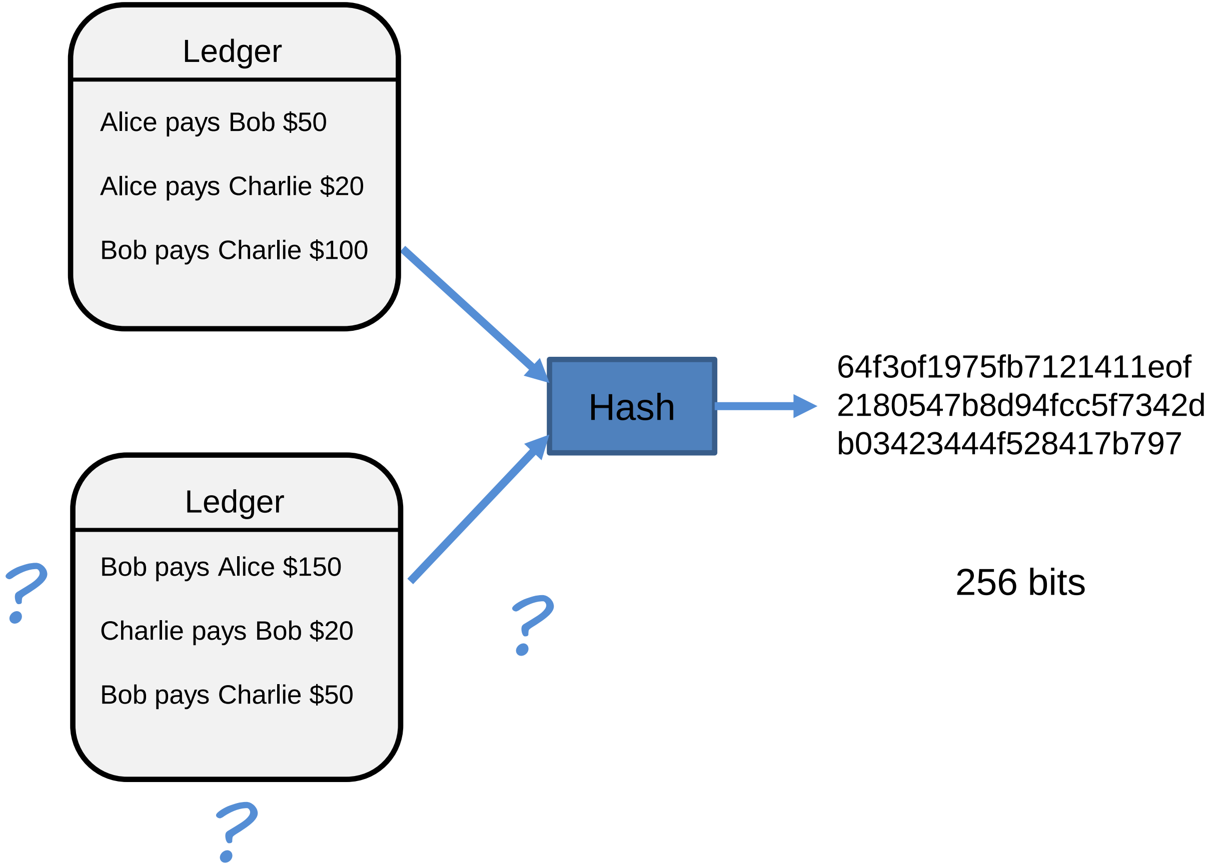
Collision resistance



256 bits

Property 1

Collision resistance



Property 1

Collision resistance

Ledger

Alice pays Bob $50

Alice pays Charlie $20

Bob pays Charlie $100

Ledger

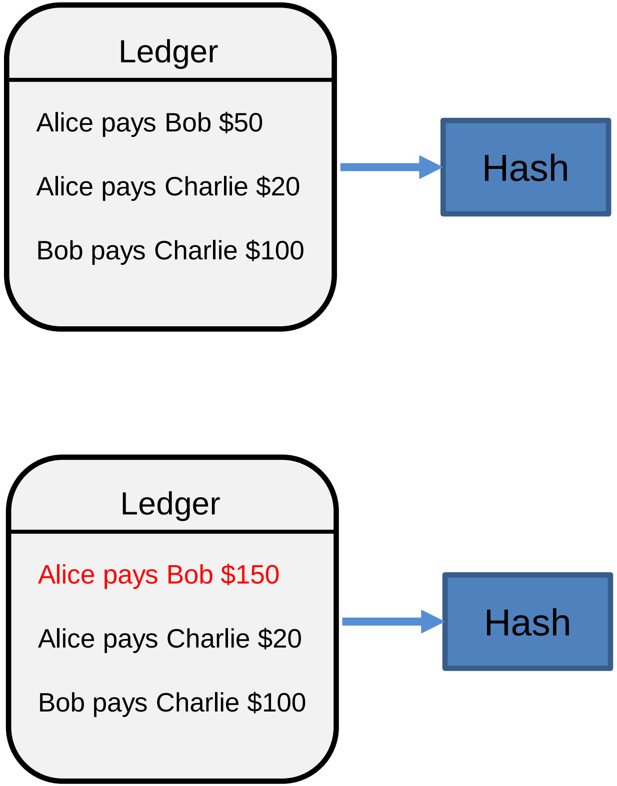
Alice pays Bob $150

Alice pays Charlie $20

Bob pays Charlie $100

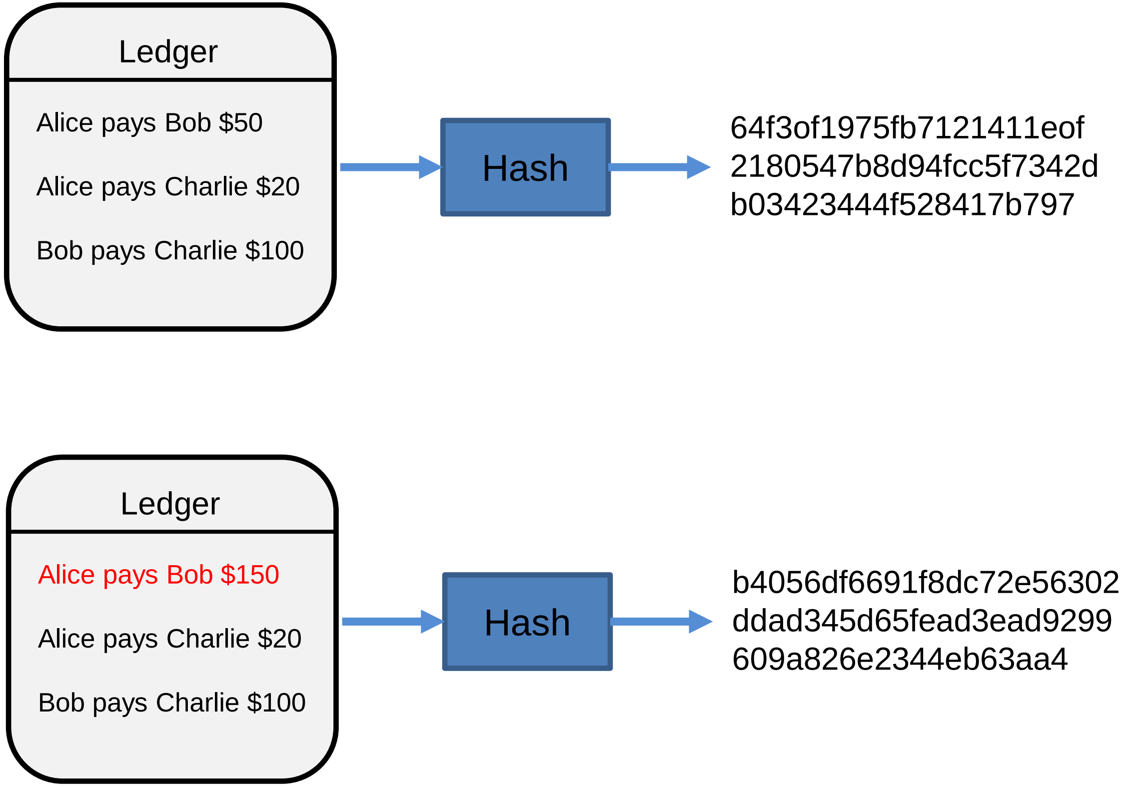
Property 1

Collision resistance



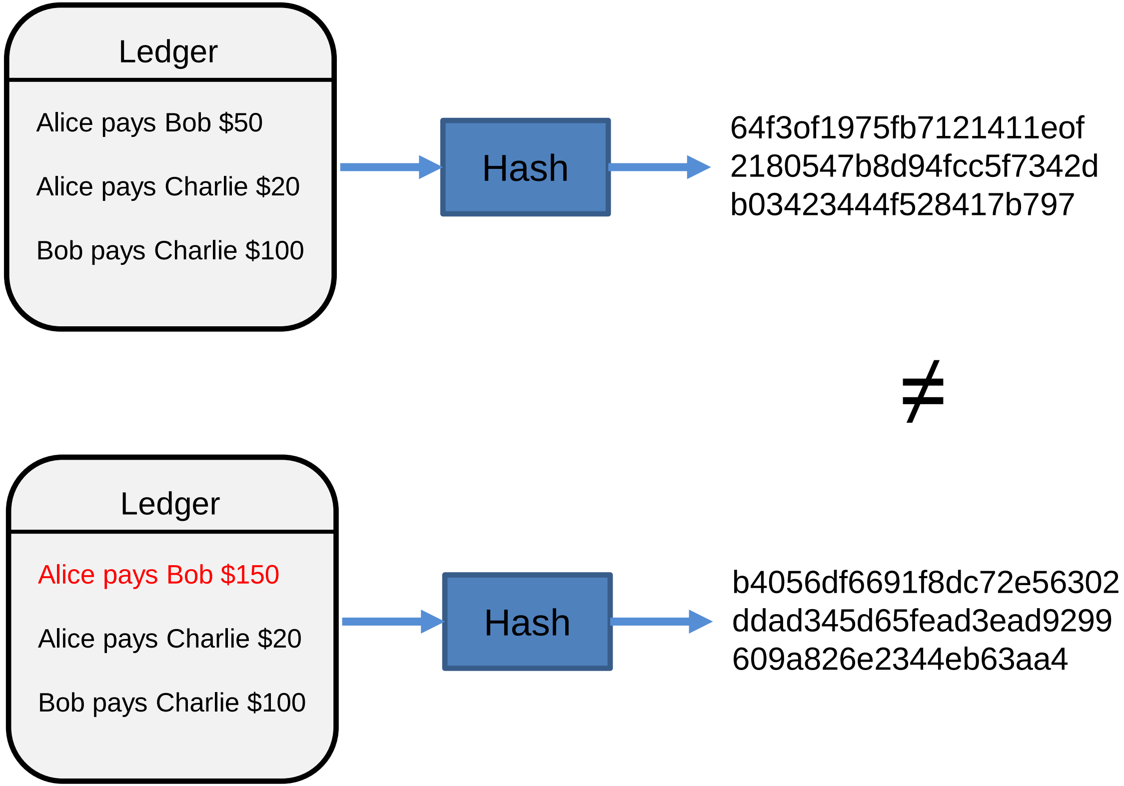
Property 1

Collision resistance



Property 1

Collision resistance



Property 1

Collision resistance

Alice

Ledger

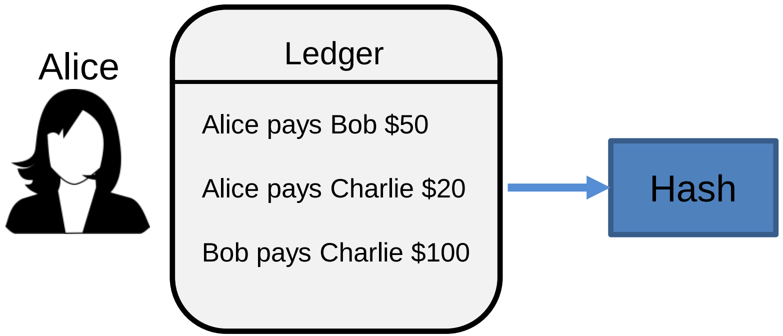
Alice pays Bob $50

Alice pays Charlie $20

Bob pays Charlie $100

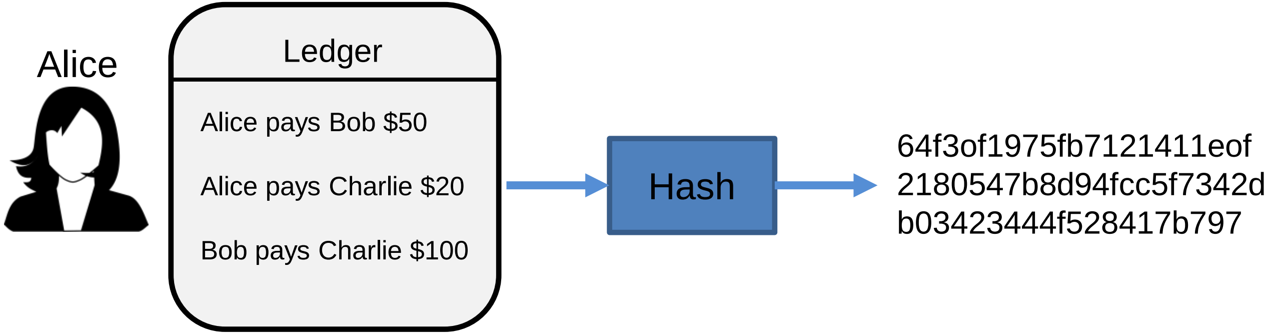
Property 1

Collision resistance



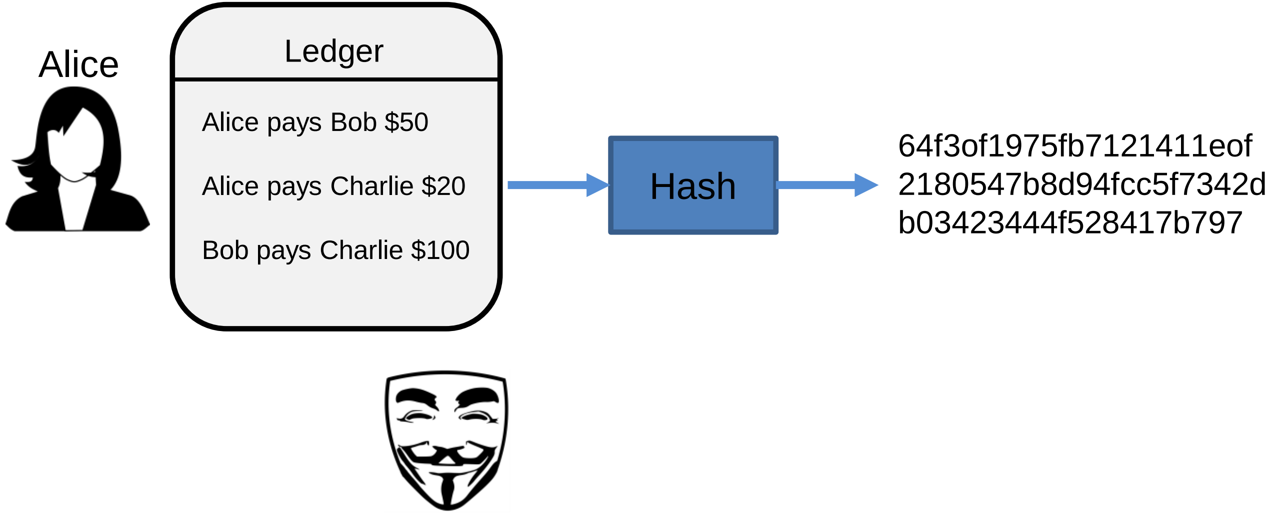
Property 1

Collision resistance



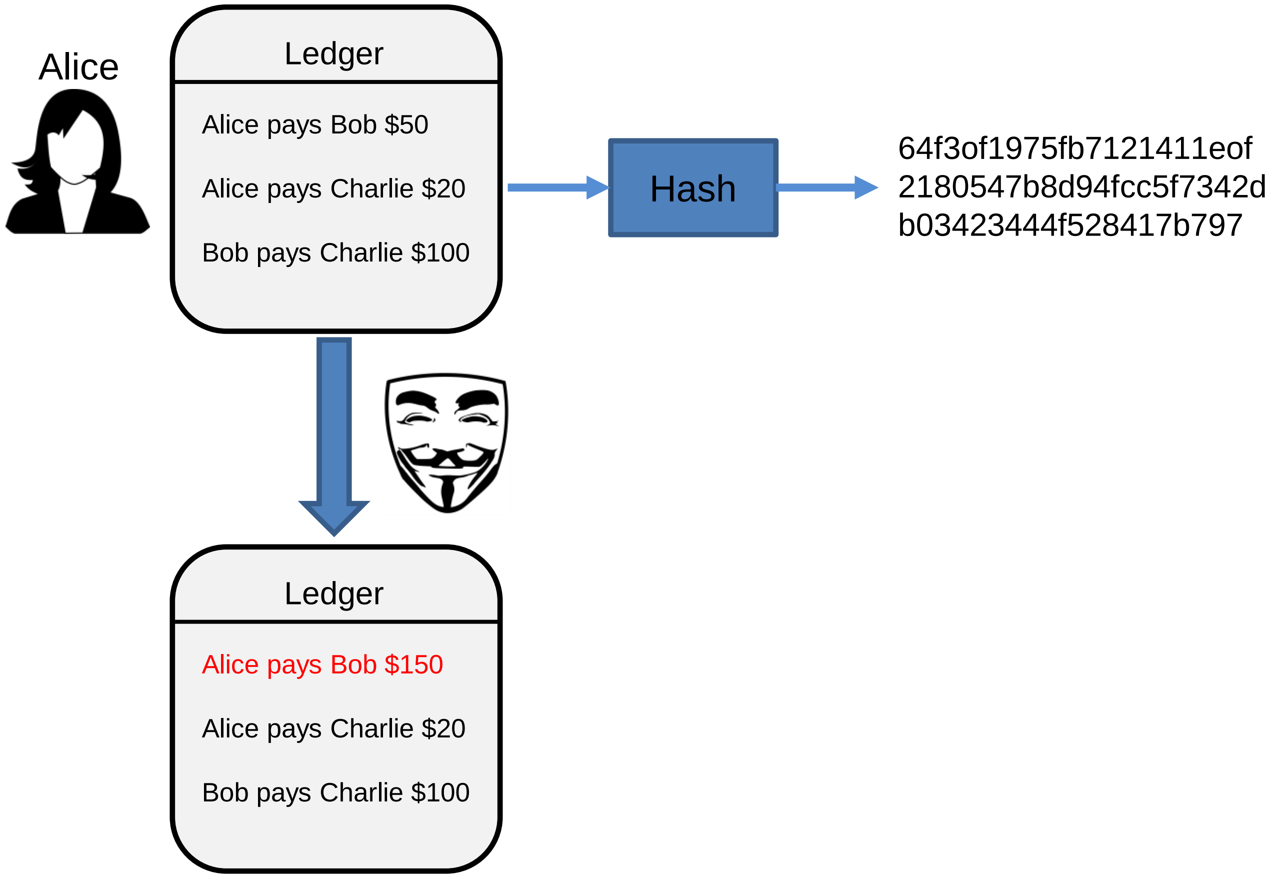
Property 1

Collision resistance



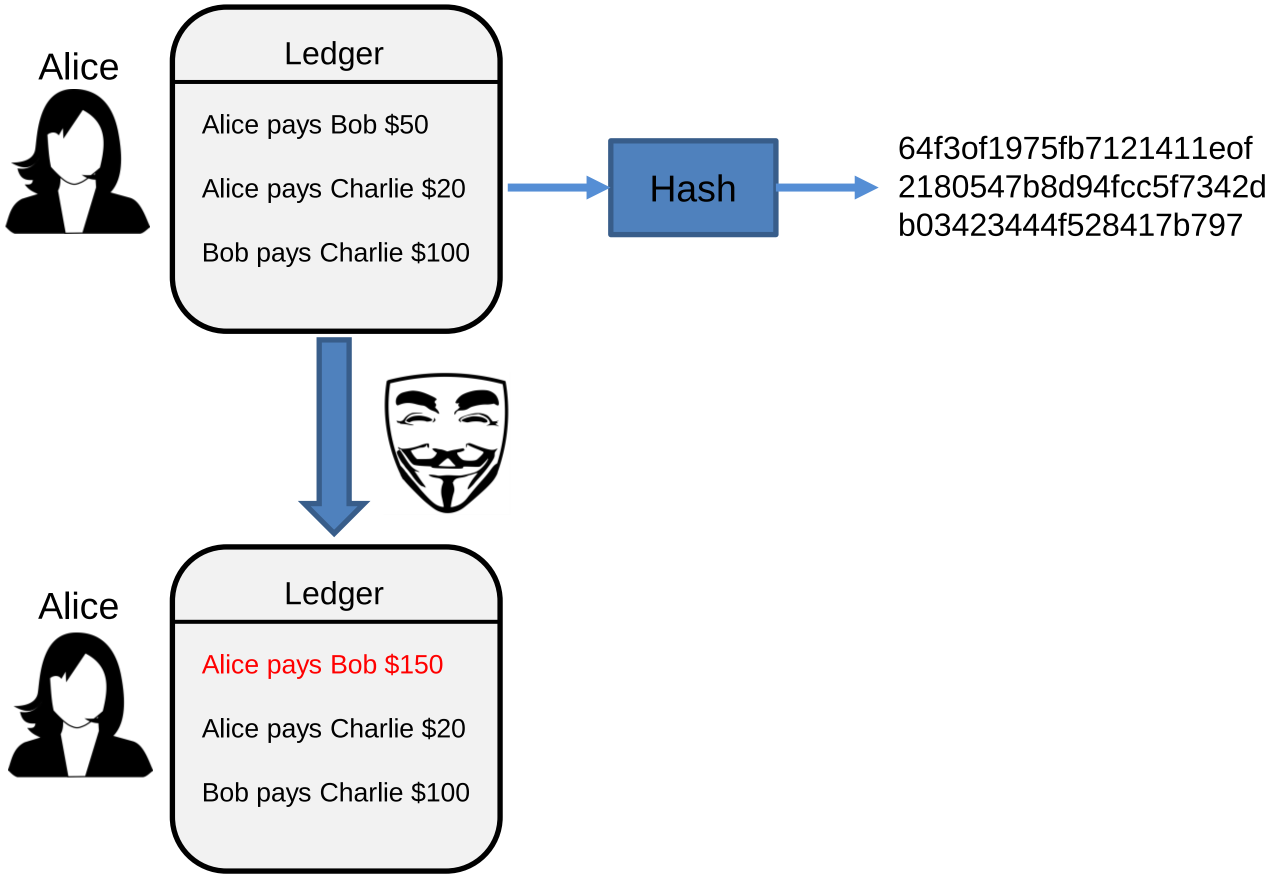
Property 1

Collision resistance



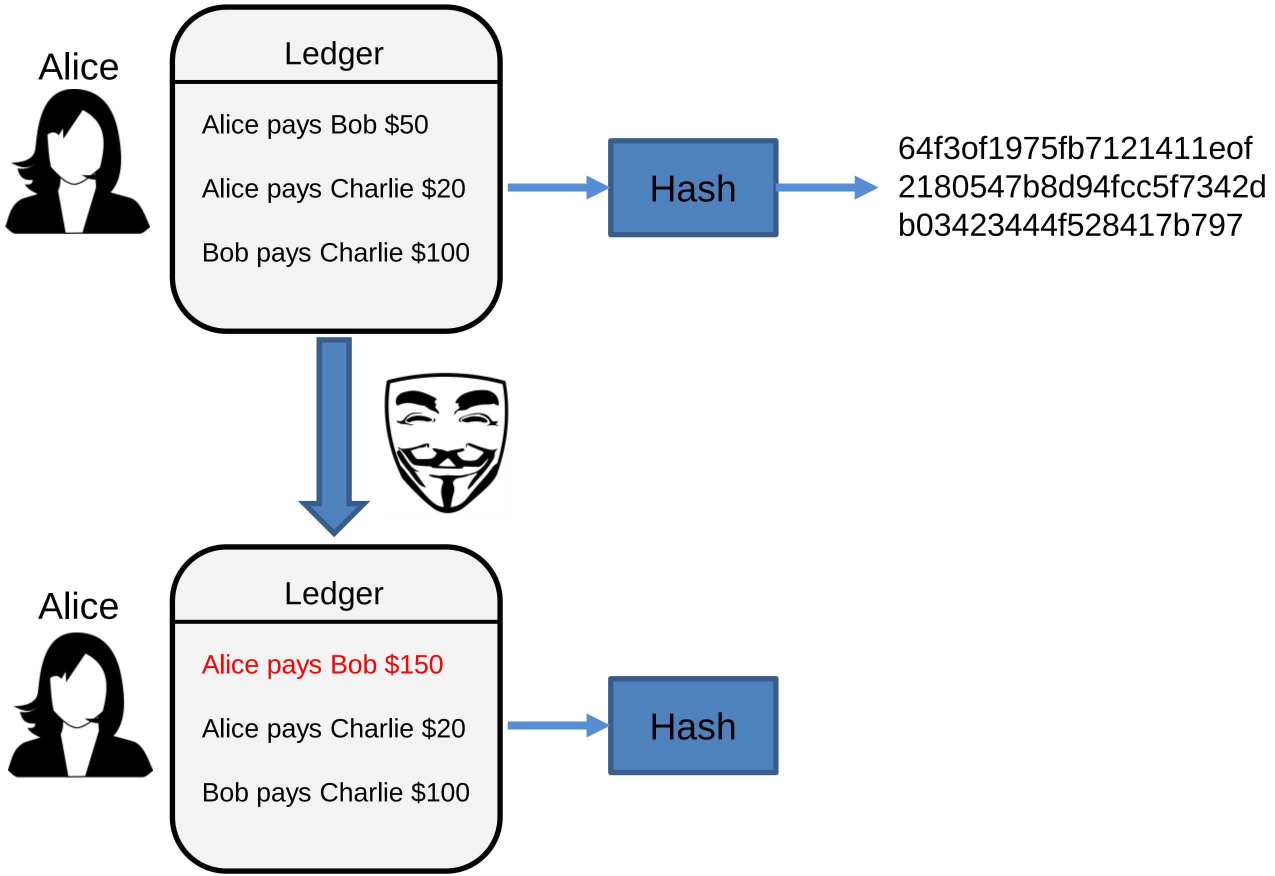
Property 1

Collision resistance



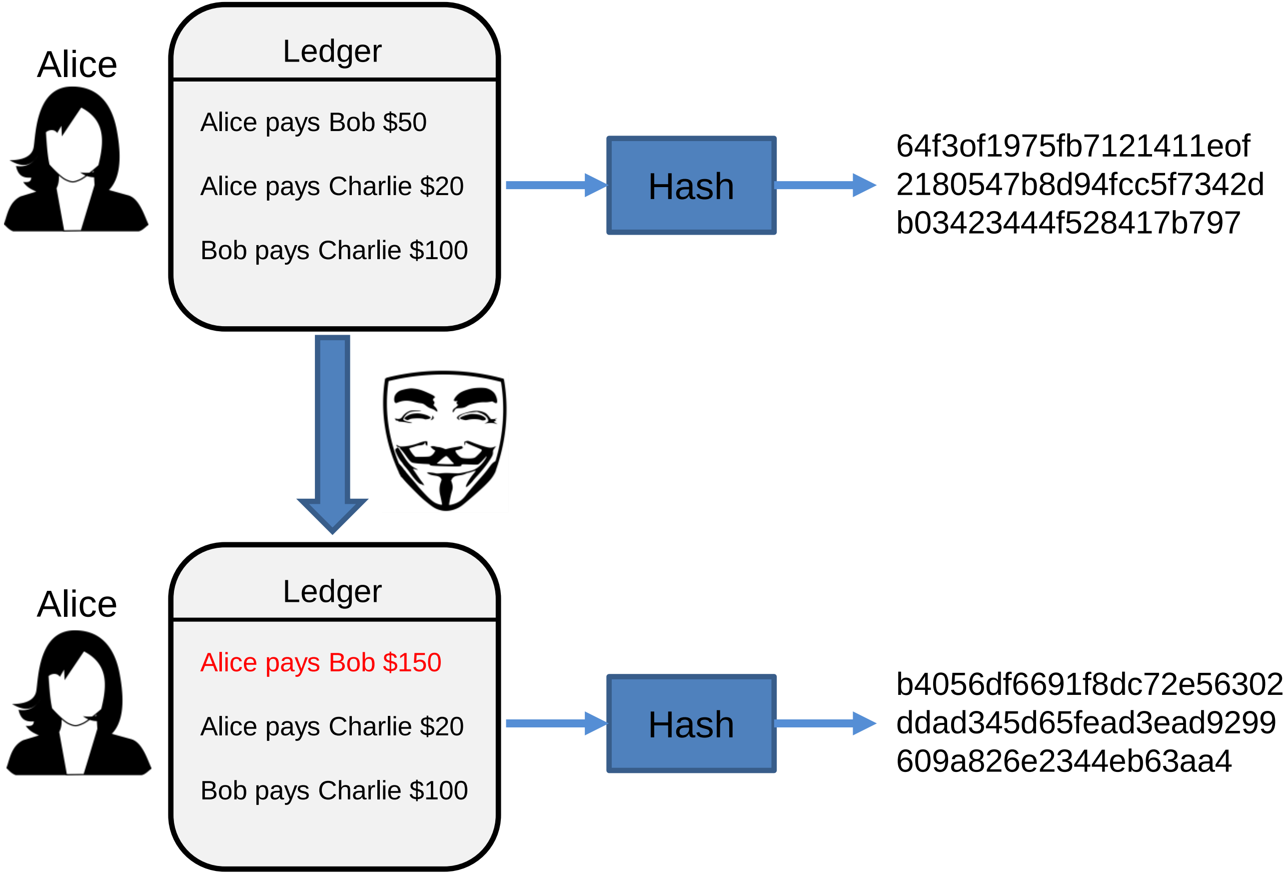
Property 1

Collision resistance



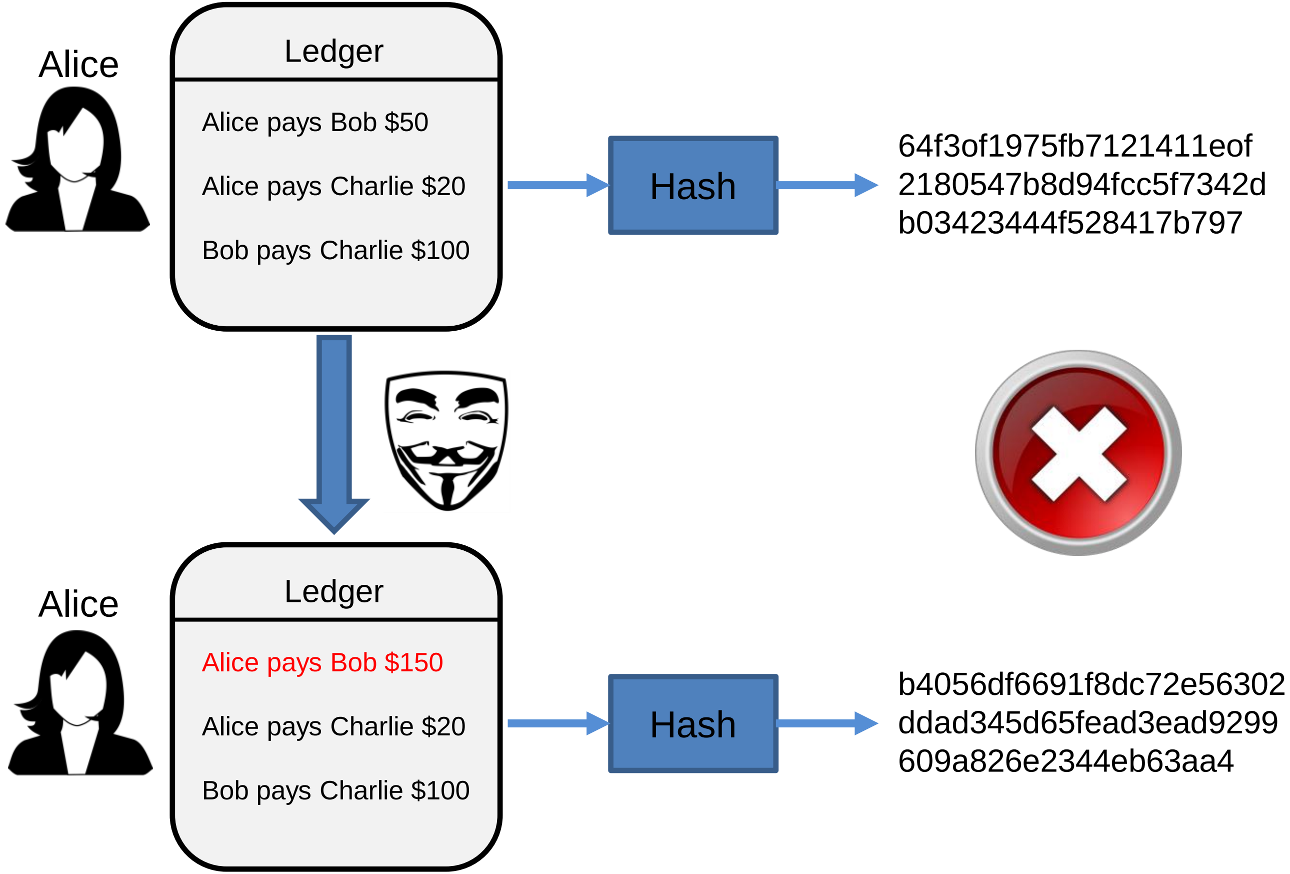
Property 1

Collision resistance

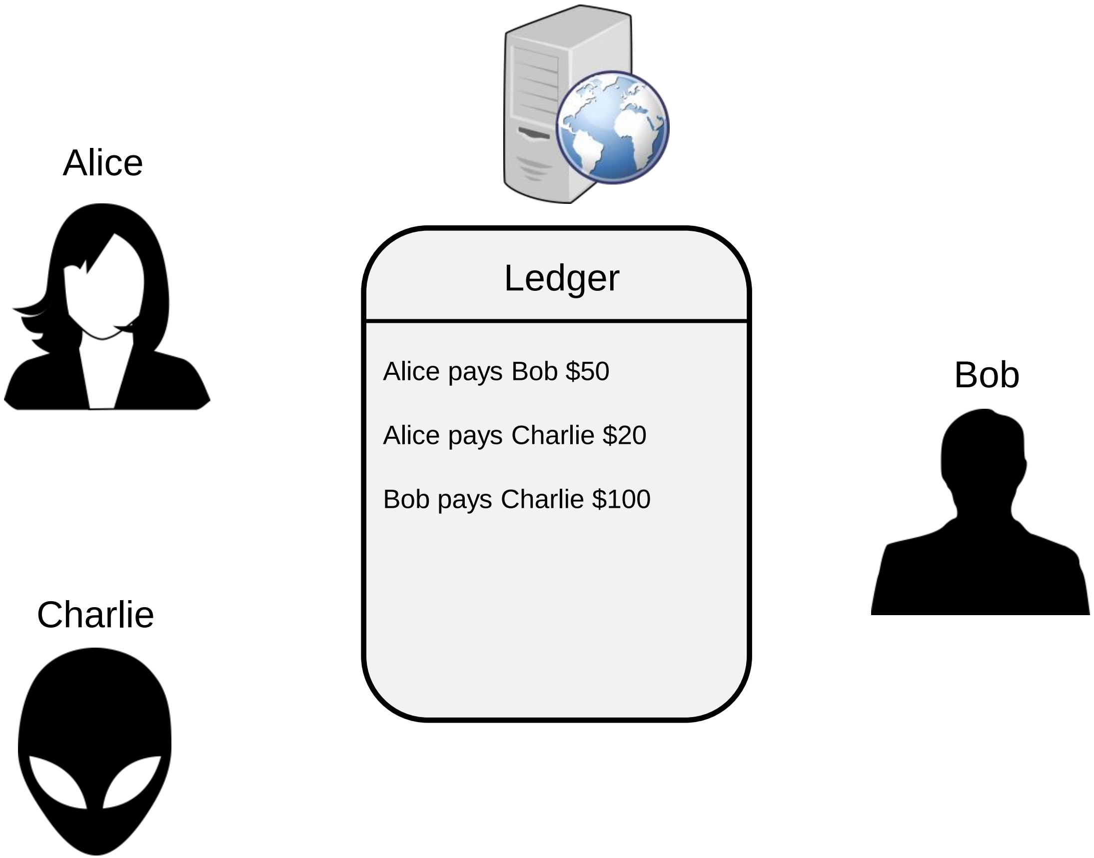


Property 1

Collision resistance

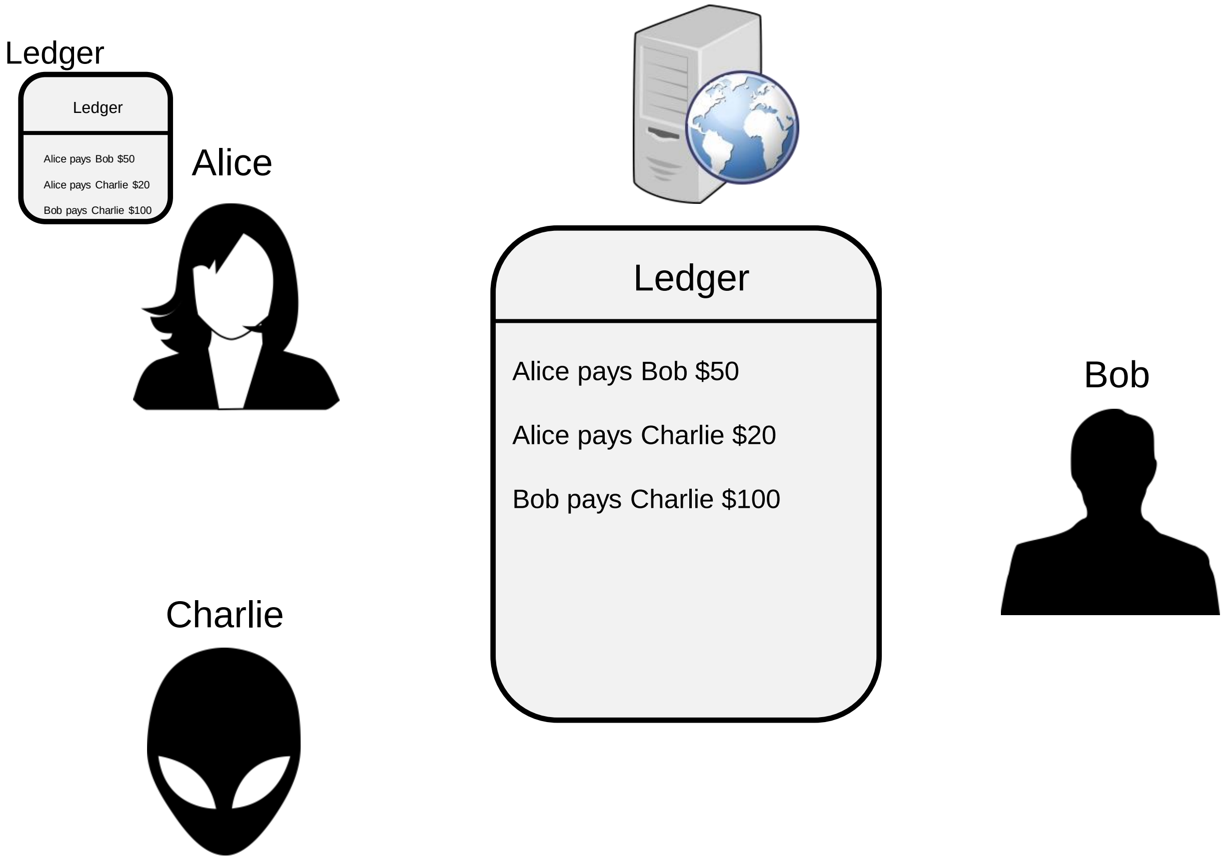


Problem 1 of e-Kuna



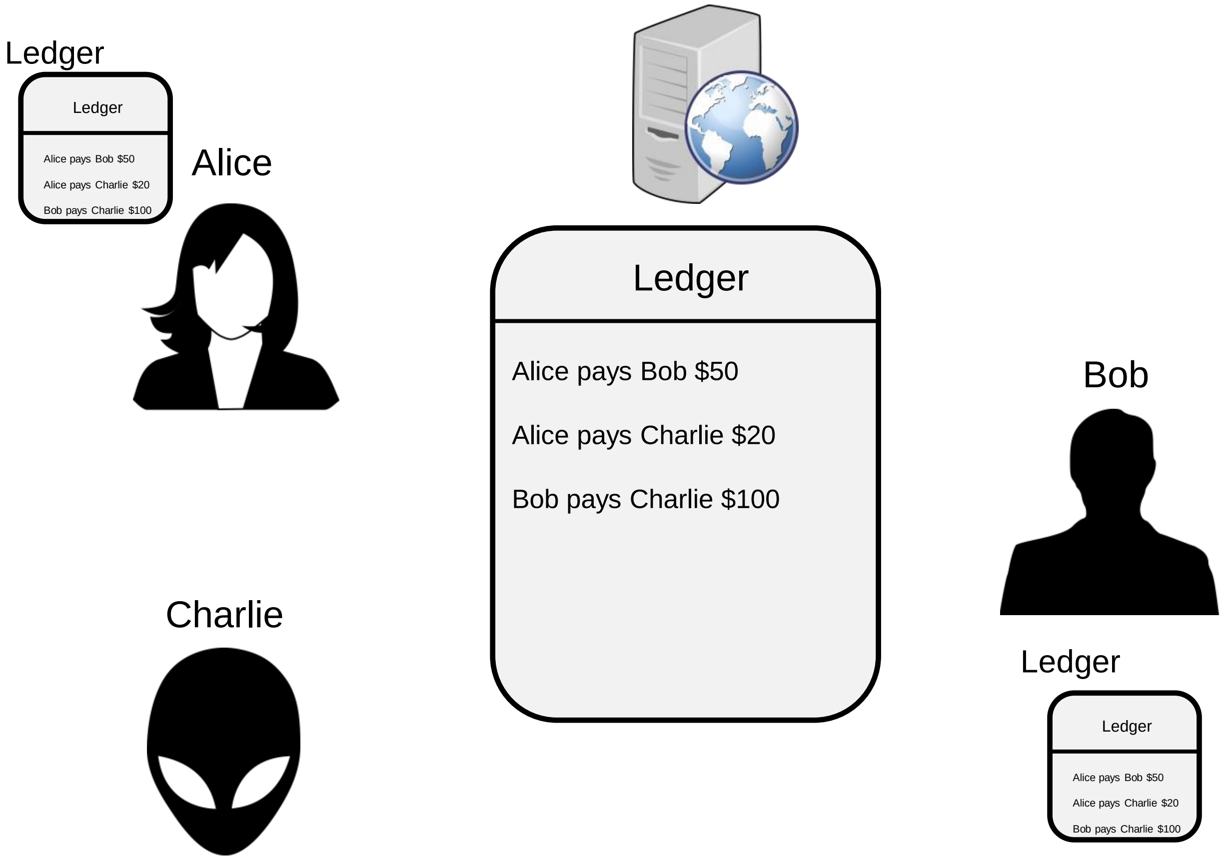
Problem 1 of e-Kuna

Ledger can be big



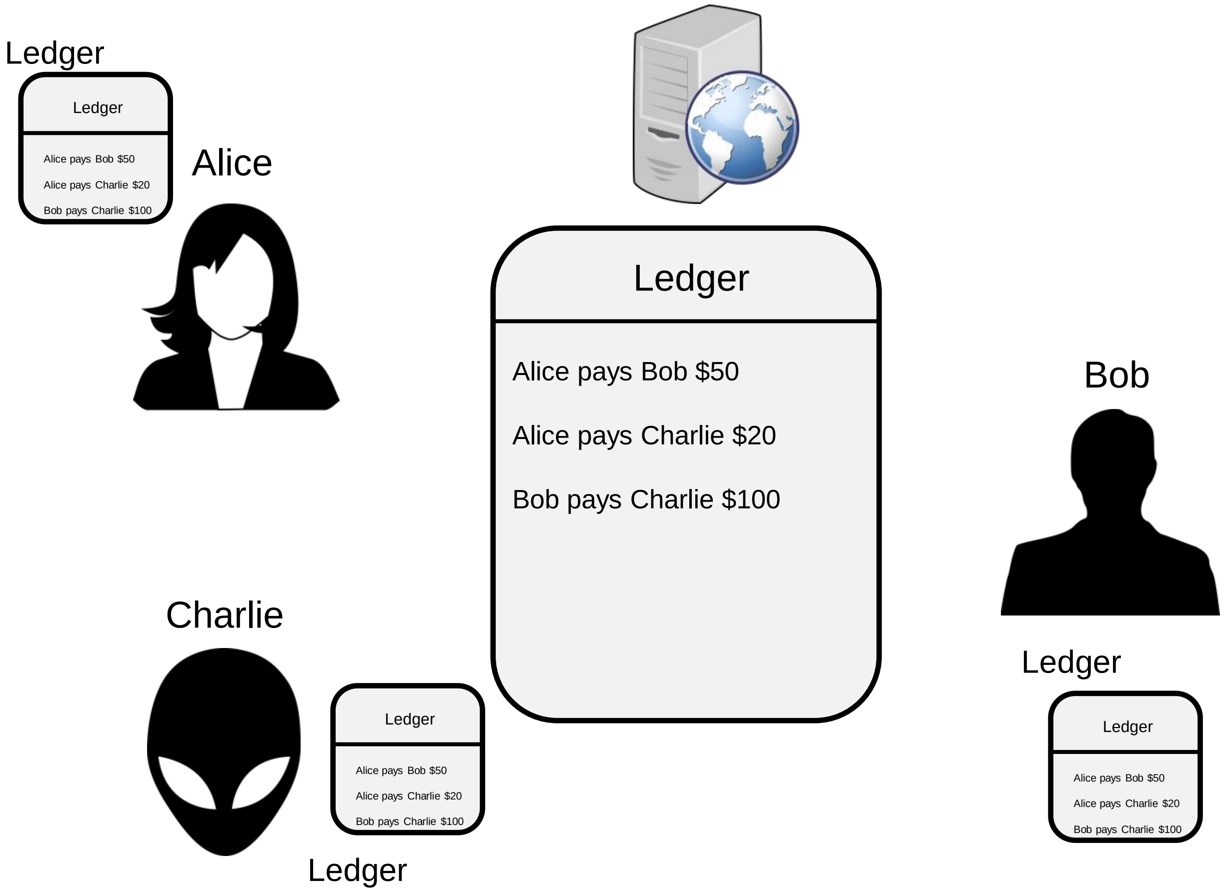
Problem 1 of e-Kuna

Ledger can be big



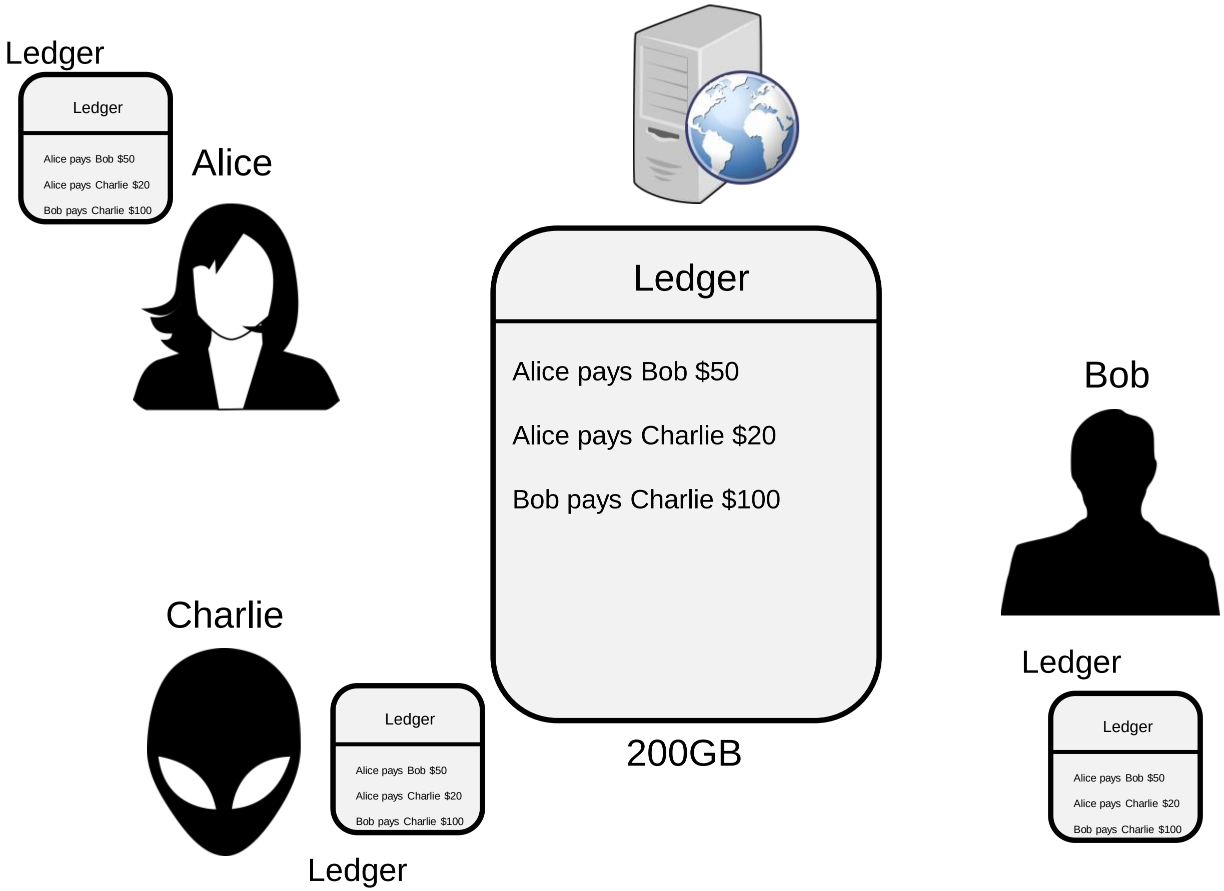
Problem 1 of e-Kuna

Ledger can be big



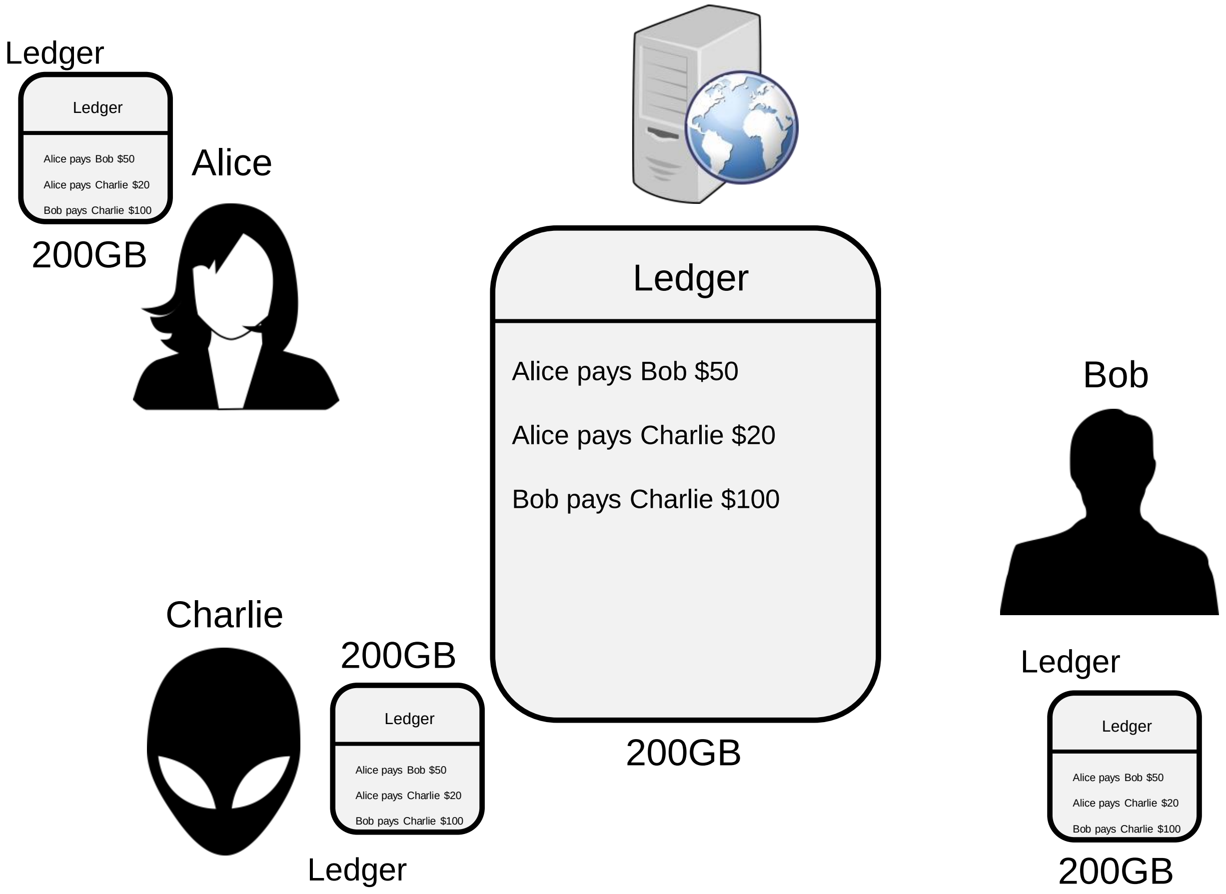
Problem 1 of e-Kuna

Ledger can be big



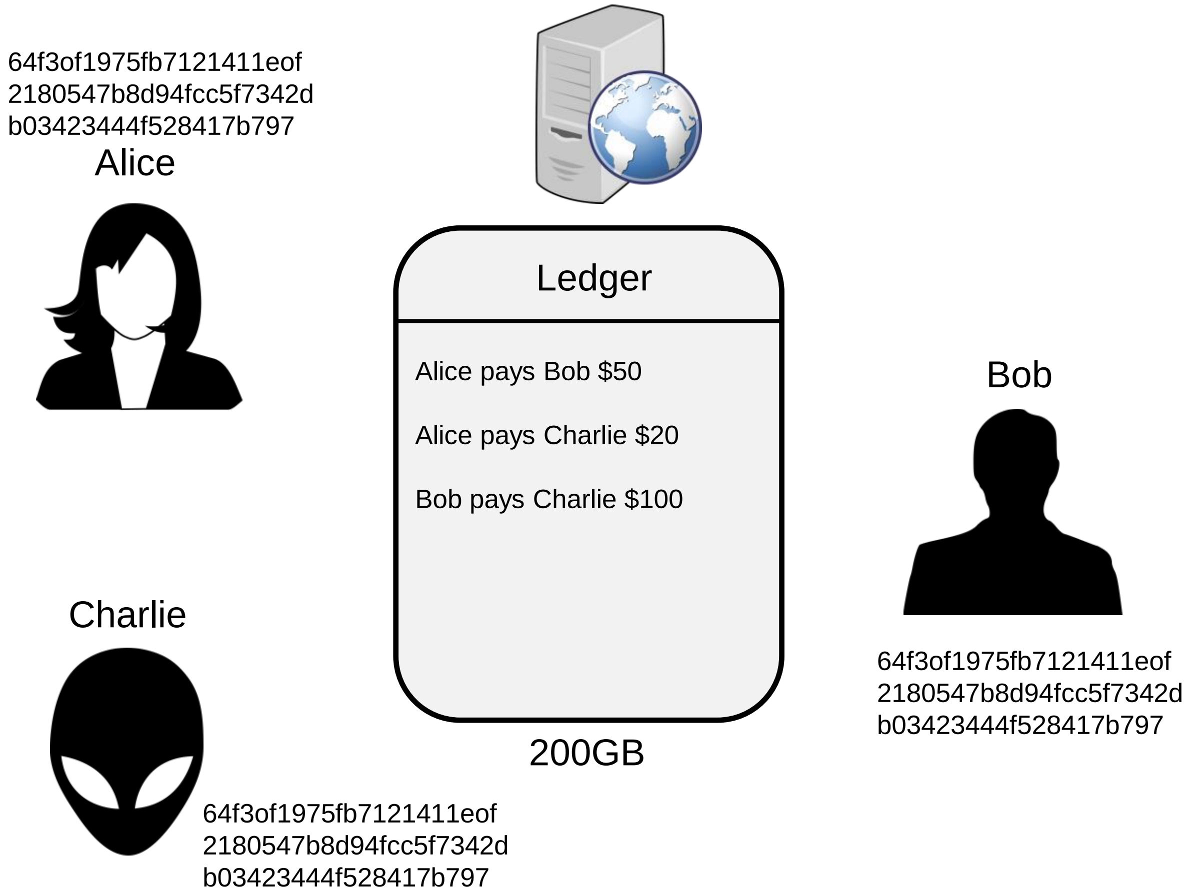
Problem 1 of e-Kuna

Ledger can be big



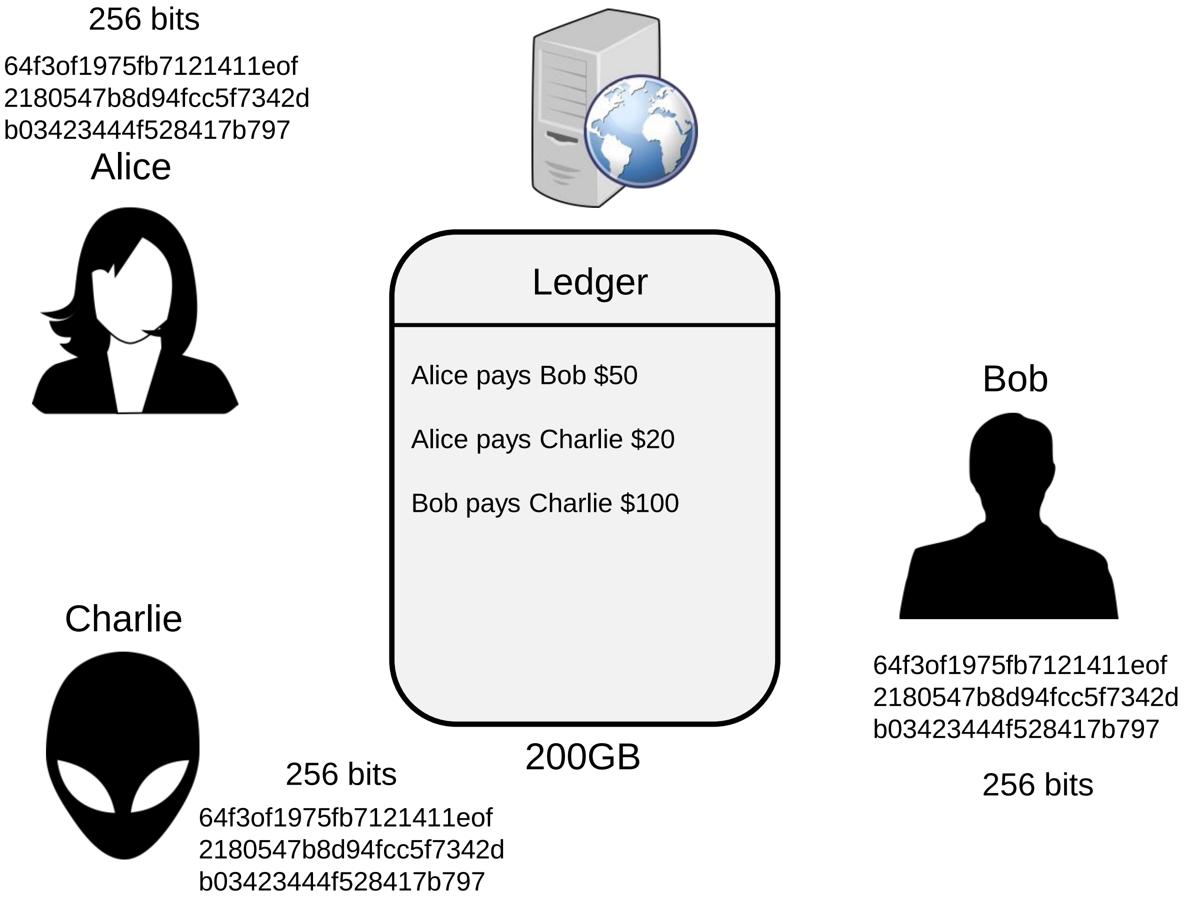
Problem 1 of e-Kuna

Ledger can be big



Problem 1 of e-Kuna

Ledger can be big



Property 1

Collision resistance

Output of H is 256 bits. How to find collisions?

- Compute hashes of 0 till $2^{256} + 1$

Question: assuming we can process 1000000 hashes/sec how long does it take?

Property 1

Collision resistance

Using the birthday paradox

- We can reduce the number to $2^{130} + 1$
- To have a 99.8% probability of a collision (blackboard)
- Is this better?

Bitcoin: 200.000.000 Th/sec (2022) and still no collisions:

- Expected time ~ 10^{11} years
- Age of universe ~ 1.3×10^{10} years

What is a cryptographic hash function?

Hash function with some security properties:

- Collision resistance
- **Hiding (hides the input)**
- Puzzle friendliness (for mining)

Property 2

Hiding

A hash function H has the **hiding** property if:

- Given $y = H(x)$ (but not x)
- It is not feasible to find the x

Property 2

Hiding

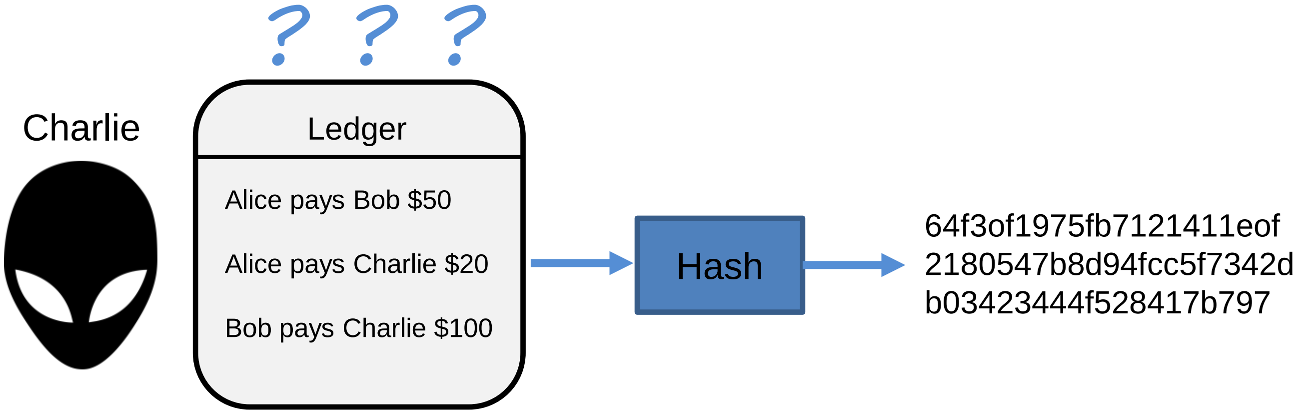
Charlie

Hash

64f3of1975fb7121411eof
2180547b8d94fcc5f7342d
b03423444f528417b797

Property 2

Hiding



Property 2

Hiding

A hash function H has the **hiding** property if:

- Given $y = H(x)$ (but not x)
- It is not feasible to find the x

If the set of inputs is small will not work (coin flip)!

Property 2

Hiding

A hash function H has the **hiding** property if:

- Given $y = H(r \mid\mid x)$
- *Where r is a random element from a huge input set*
- It is not feasible to find the x

r in bitcoin is from the (integer) interval $[0,\ldots,2^{256}]$

Use of hiding: a commitment

Simulate putting a sealed enevelope on a table:

- I write a message
- I seal it in an envelope
- I place the envelope on a table for everyone to see
- When opened later, everyone can see my message

Digital commitment protocol

Two algorithms:

- *com := commit(msg,nonce)*
- *verify(com,msg,nonce)* returns *true/false*

nonce is randomply chose (always)

Digital commitment protocol

Properties of the two algorithms:

- **Hiding**: given *com* it is not feasible to find *msg*
- **Binding**: it is not feasible to find *(msg,nonce)* and *(msg',nonce’) s*.t. *commit(msg,nonce) == commit(msg',nonce')*

Digital commitment protocol

$commit(msg, nonce) := H(msg \mid\mid nonce)$:

- **Hiding**: hiding of *H*
- **Binding**: collision resistance of *H*

What is a cryptographic hash function?

Hash function with some security properties:

- Collision resistance
- Hiding (hides the input)
- **Puzzle friendliness (for mining)**

Property 3

Puzzle friendliness

A hash function H is **puzzle friendly** if:

- Given an n-bit output y of H
- And given any k (from a large set, uniformly distributed)
- It is not feasible to find x s.t. $H(k \,||\, x) = y$
- In time less than $O(2^n)$

Property 3

Puzzle friendliness

Charlie

Property 3

Puzzle friendliness

Charlie

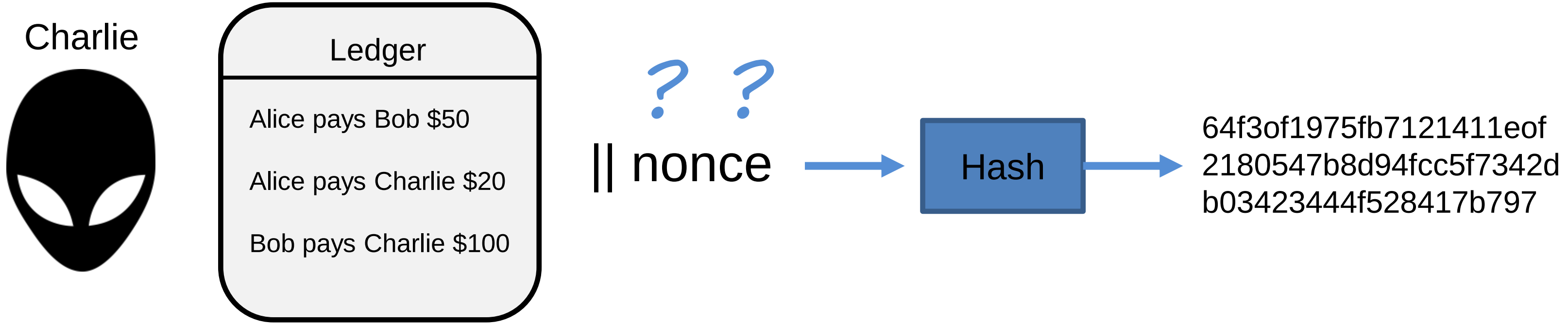
Hash → 64f3of1975fb7121411eof
2180547b8d94fcc5f7342d
b03423444f528417b797

Property 3

Puzzle friendliness

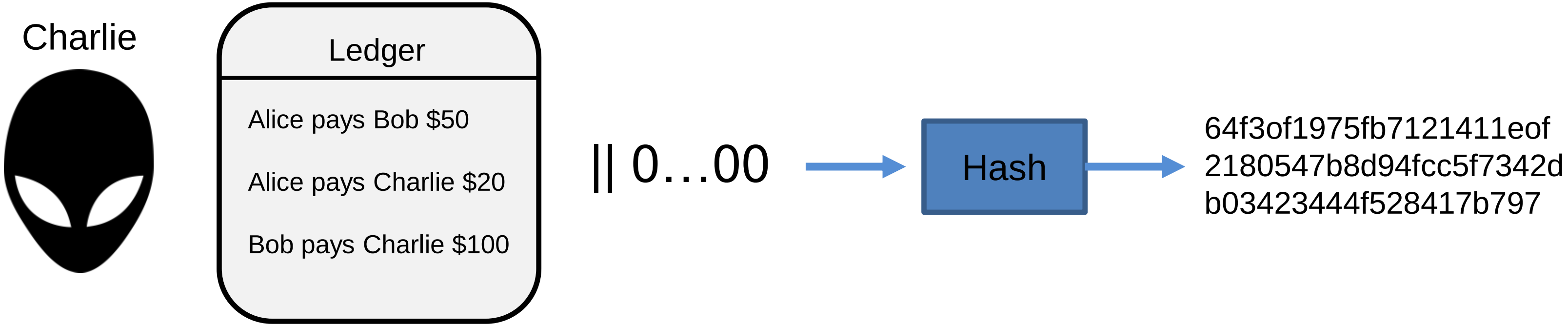
Property 3

Puzzle friendliness



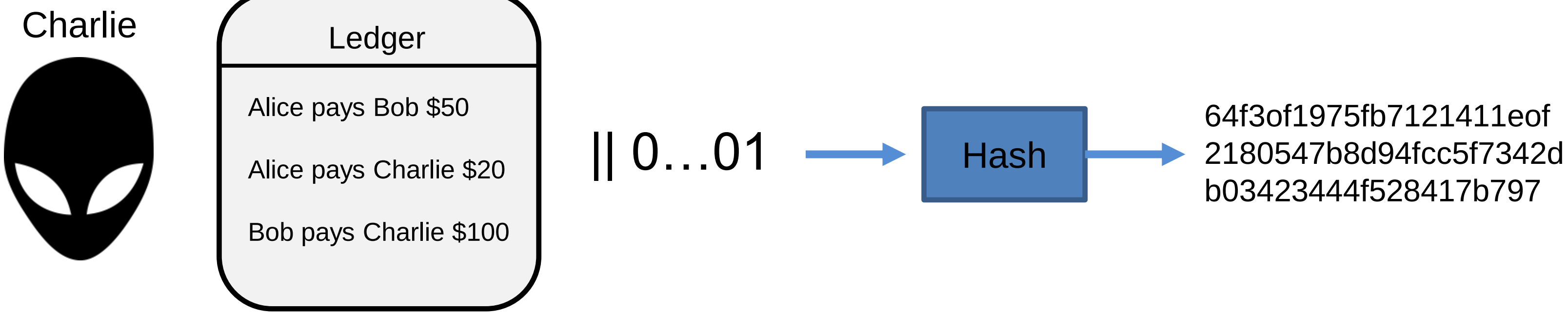
Property 3

Puzzle friendliness



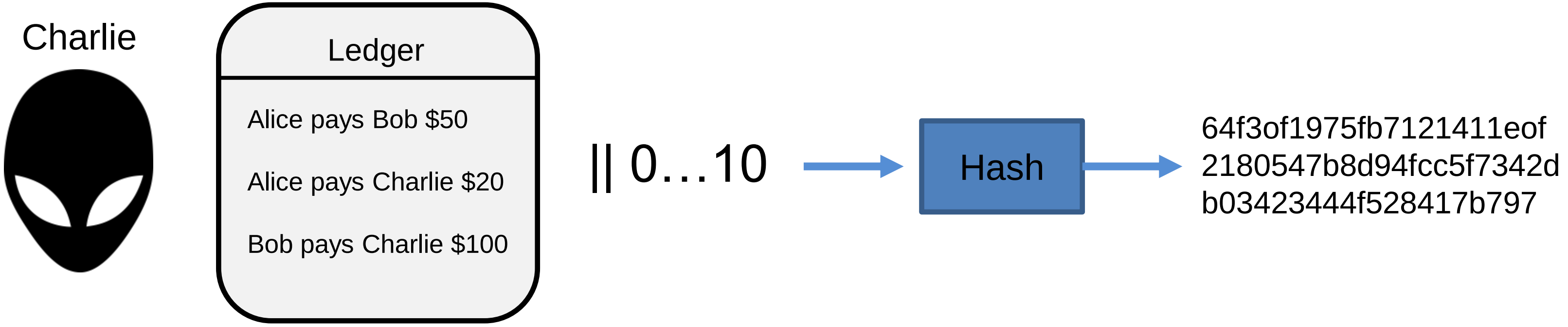
Property 3

Puzzle friendliness



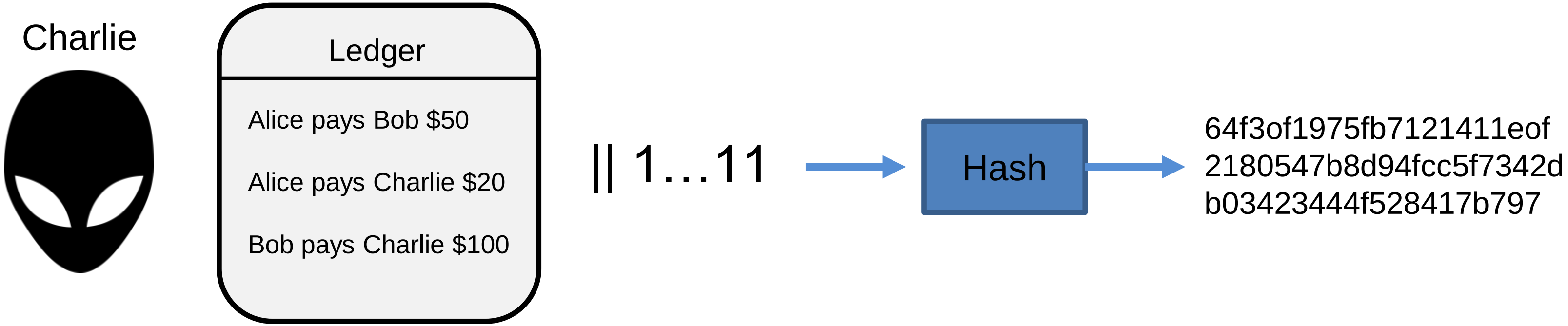
Property 3

Puzzle friendliness



Property 3

Puzzle friendliness



Mining

Search puzzle

A **search puzzle** consists of:

- A cryptographic hash function H
- A puzzle ID
- A target set Y

A **solution for the search puzzle** is a value x s.t

- $H(ID \,||\, x)$ belongs to Y

Mining

Search puzzle

A **solution for the search puzzle** is a value x s.t

- *H(ID || x)* belongs to *Y*

Puzzle friendly: any *x* is equally probable as a solution for the search puzzle

Mining

Search puzzle

Difficulty of the search puzzle: the size of Y

- $Y = [0,\ldots,2^{256}]$ in H with 256 bits is trivial
- $Y = \{y\}$ maximal difficulty
- All the intermediate sizes

That is how mining difficulty in Bitcoin is controlled!

Bitcoin's hash function

SHA-256

NIST/NSA standard

General properties of hash functions:

- Input of fixed size – *m*
- Output of fixed size – *n*
- Transformation Merkle-Dagmår

Bitcoin’s hash function

SHA-256

Transformation Merkle-Dagmår:

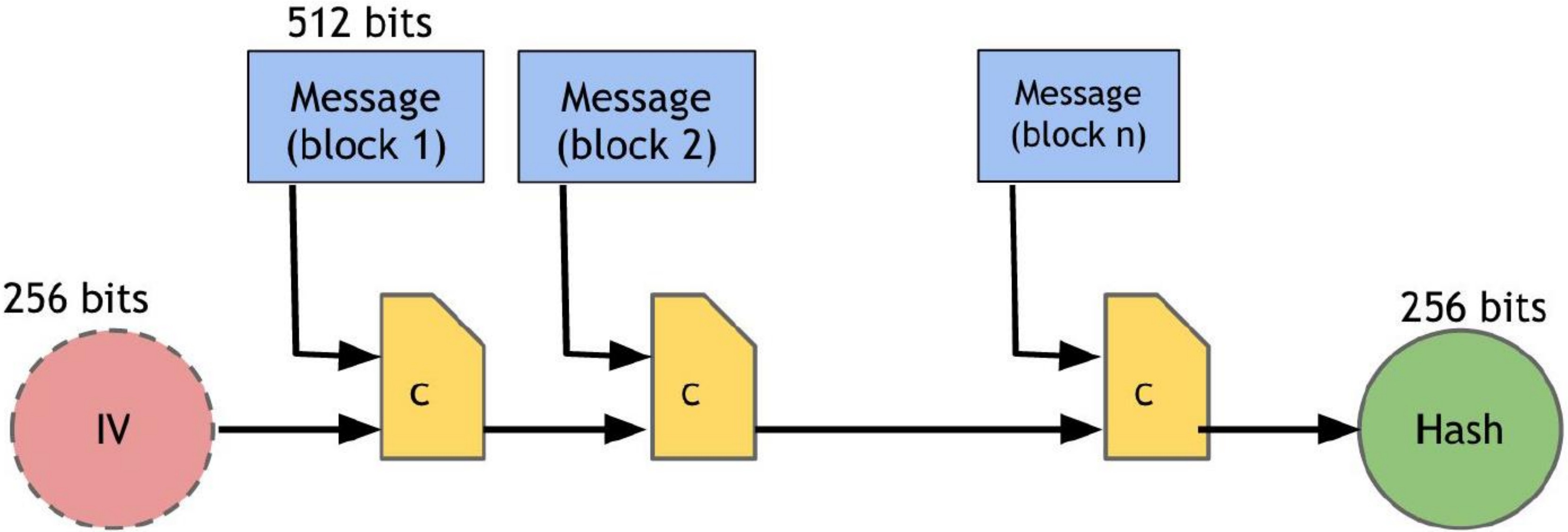
- Divide the input in blocks of size m-n
- Process block by block together with the previous output
- Use an initialisation vector (IV) for the first round

SHA-256:

- Input of size 768 bits
- Blocks of size 512 bits
- Output 256 bits

Bitcoin's hash function

SHA-256



Practice time

SHA-256

(assuming Python 3+)

(https://www.python.org/)

We will need: hashlib (probably already installed)

Hash functions in Python

Computing a hash

hash.py

```python
import hashlib

# normal sha256
def hash(message):
    # will return bytes of the created sha256 object
    return hashlib.sha256(message).digest()


# double sha256
def hash256(message):
    '''two rounds of sha256'''
    # will return bytes of the created sha256 object
    return hashlib.sha256(hashlib.sha256(message).digest()).digest()

```

Hash functions in Python

Computing a hash

hash_test.py

```python
from hash import hash

data1 = b'Cryptocurrency'
hash1 = hash(data1)
print("Bytes: ",hash1)
hash1_hex = hash1.hex()
print("Hex:    ",hash1_hex)

data2 = b'cryptocurrency'
hash2 = hash(data2)
print("Bytes: ",hash2)
hash2_hex = hash2.hex()
print("Hex:    ",hash2_hex)
```

```
Bytes:  b"n\xc6\x0f\xe3\x90(\x88~\x7f\xe9\xc4\xb0%TWH\x95<'QPs\xbf\x9a\xd1|\xebT\x17\xa4\x07\xd7"
Hex:    6ec60fe39028887e7fe9c4b025545748953c27515073bf9ad17ceb5417a407d7
Bytes:  b'\xa8\x12F\x02>?lag\xa0\x8b\xa2$@\x90&\xf8\x8b\xb8\xe9\x8e\xd1C\x1c\xd5<\xb6:2\x8cn\x84'
Hex:    a81246023e3f6c6167a08ba224409026f88bb8e98ed1431cd53cb63a328c6e84
[Finished in 72ms]
```

Exercise

SHA-256

Program a mining function:

- Input: puzzle *ID* (as a SHA-256 hash/output of SHA-256)
- Input: target set (given as y in $[0,\ldots,2^{256}]$)
- Target: $\{ z \text{ in } [0,\ldots,2^{256}] : z < y \}$ (how to check this quickly?)
- The function searches for a *nonce* s.t. $SHA256(\, ID \,||\, nonce \,) < y$

Think of different ways of searching for the nonce!!!

What is the best startegy using only one machine?

Exercise

In Bitcoin

Bitcoin mining uses the following parameters:

- Input: puzzle *ID* (as a SHA-256 hash)
- Input: target set (*y* in $[0,\ldots,2^{256}-1]$)
- Target: $\{ z \text{ en } [0,\ldots,2^{256}-1] : z < y \}$
- Module searces for a *nonce* s.t. $SHA256(ID \,||\, nonce) < y$
- But in Bitcoin *nonce* comes from $[0,\ldots,2^{32}-1]$

There are puzzles without a solution!!!

How does Bitcoin deal with this???

Exercise

Mining strategies

Let’s use these parameters:

- Input: puzzle *ID* (as a SHA-256 hash)
- Input: target set (y in $[0,\dots,2^{256}-1]$)
- Target: $\{ z \text{ en } [0,\dots,2^{256}-1] : z < y \}$
- Module searces for a *nonce* s.t. $SHA256(\ ID \ || \ nonce\ ) < y$
- But in Bitcoin *nonce* comes from $[0,\dots,2^{32}-1]$

Try different strategies when searching for a nonce.

Detect when there is no solution!!!! (o estimate it)

Exercise

Mining strategies

nonce belongs to *[0,…,2^{23} -1] (cca 40 sconds on my computer):*

- Mine in ascending order
- Mine in descending order
- Mine randomly

Let us try the three strategies with different targets!!!

Exercise

Mining pools

Mining pools:

- Divide the search space amongst the participants in the pool
- Each member mines only a certain range of inputs

Competetition between pools for different targets:

- Play with the difficulty
- *nonce* in $[0,\ldots,2^{22} - 1]$