how to:
create your own blockchain

pramod subramanyan

spramod@cse.iitk.ac.in

26-December-2018

what is a block chain?

> “an open, distributed ledger that can record transactions between two parties efficiently and in a verifiable and permanent way”

Marco Iansiti and Karim R. Lakhani
“The Truth about Blockchain”,
Jan-Feb 2017, Harvard Business Review

in the rest of this presentation

we will learn how to

1. build a ledger of transactions
2. make it verifiable and permanent

what is a ledger?

account holder	balance
alice	100
bob	200
carol	300
dan	400

let’s start with a table of account balances

not actually a ledger

a ledger records transactions

debit account	credit account	amount
initial deposit	alice	100
initial deposit	bob	200
initial deposit	carol	300
initial deposit	dan	400

this is now a ledger

now suppose bob pays alice ₿100

debit account	credit account	amount
initial deposit	alice	100
initial deposit	bob	200
initial deposit	carol	300
initial deposit	dan	400
bob	**alice**	**100**

and alice pays carol ₿125

debit account	credit account	amount
initial deposit	alice	100
initial deposit	bob	200
initial deposit	carol	300
initial deposit	dan	400
bob	alice	100
alice	**carol**	**125**

and so on ..

debit account	credit account	amount
initial deposit	alice	100
initial deposit	bob	200
initial depo		00
initial depo		00
bob	alice	100
alice	carol	125
…	…	…

all transactions are recorded by appending to the ledger

ledger to account balances?

sender	receiver	amount
initial deposit	alice	100
initial deposit	bob	200
initial deposit	carol	300
initial deposit	dan	400
bob	alice	100
alice	carol	125

account	amount
alice	100+100-125=75
bob	200-100=100
carol	300+125=425
dan	400

but not all transactions are valid

sender	receiver	amount
initial deposit	alice	100
initial deposit	bob	200
initial deposit	carol	300
initial deposit	dan	400
bob	alice	100
alice	carol	125
bob	**dan**	**200**

account	amount
alice	75
bob	100
carol	425
dan	400

bob doesn’t have ฿200
in his account

definition: transaction validity (v1)

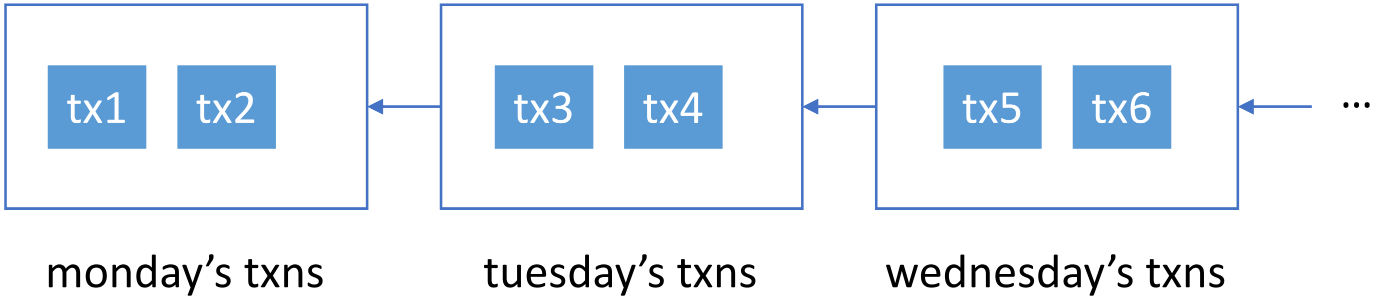
transaction valid if sender’s balance is >= amount being sent to receiver

definition: ledger validity (v1)

ledger valid if all transactions in it are valid

that is, every sender has the appropriate balance to conduct every transaction

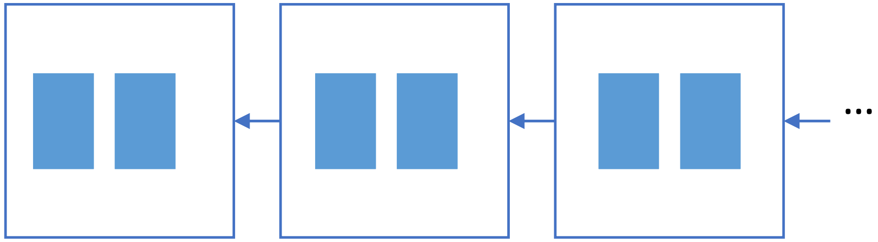
blockchain: ledger of transactions



- each block has txns from specific time period
- blockchain = linked list of blocks

let’s see some code

```
class txn_t {
  account_t sender;
  account_t receiver;
  uint64_t amount;
};
```



```
class block_t {
  std::vector<txn_t*> txns;
  block_t* prev_block;
};
```

```
std::list<block_t*> blockchain;
```

`std::vector<T>`: resizeable array

- insertion

```
vec.push_back(elem);
```

- size

```
vec.size()
```

- element access

```
vec[i] = blah;
```

- access to raw array inside

```
vec.data()
```

- resize to have n elements:

```
vec.resize(n);
```

- delete all elems and set size = 0

```
vec.clear();
```

- iteration

```
for (auto elem : vec) {
    // do s’th with elem;
}
```

`std::list<T>`: linked list

- insertion

```
lst.push_back(elem);
```

- size

```
lst.size()
```

- delete all elems and set size = 0

```
lst.clear();
```

- iteration (v1):

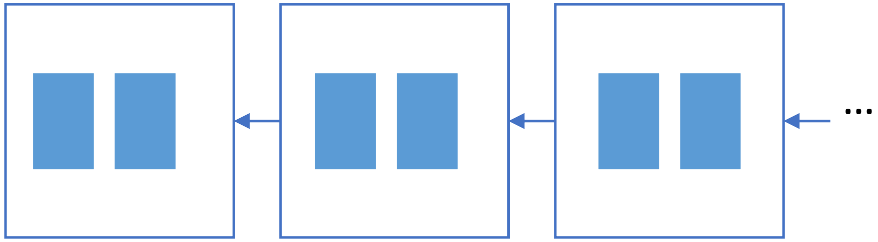
```
for (auto p = lst.begin();
     p != lst.end(); p++)
{
    // do s’th with *p;
}
```

- iteration (v2)

```
for (auto elem : vec) {
    // do s’th with elem;
}
```

in modern c++ (c++’11 and c++’14)

```
class txn_t {
  account_t sender;
  account_t receiver;
  uint64_t amount;
};
```



```
class block_t {
  std::vector<std::shared_ptr<txn_t> > txns;
  std::shared_ptr<block_t> prev_block;
};
```

```
std::list<std::shared_ptr<block_t> > blockchain;
```

use smart pointers

std::shared_ptr<T> is a reference counted smart pointer

creation:

- `shared_ptr<block_t> ptr(new block_t(…));`

deletion:

- do nothing! auto deleted when out of scope

access:

- `ptr->member` (just like a pointer)

copying:

- `shared_ptr<block_t> ptr2 = ptr;`

blockchain validation code (v1)

```
bool validate(list<shared_ptr<block_t> >& blockchain) {
  balances.clear();
  for (auto blk : blockchain) {
    for (auto tx : blk.txns) {
      if (tx.sender == INIT_DEPOSIT ||
          balances[tx.sender] >= tx.amount)
      {
        balances[tx.sender] -= tx.amount;
        balances[tx.receiver] += tx.amount;
      } else {
        return false;
      }
    }
  }
  return true;
}
```

iterate over each transaction

check balance

update balances

unordered_map<K,V>: hashtable

- insertion

```
m [key] = elem;
```

- size

```
m.size()
```

- delete all elems and set size = 0

```
m.clear();
```

- iteration:

```
for (auto p = m.begin();
          p != m.end(); p++)
{
        // use p.first (key)
        // and p.second (val)
}
```

- search

```
auto p = m.find(k);
// returns m.end() or can
// use p.first, p.second
```

set<T>: set of elements

- insertion

  ```
  m.insert(elem)
  ```

- size

  ```
  m.size()
  ```

- delete all elems and set size = 0

  ```
  m.clear();
  ```

- iteration:

  ```
  for (auto p = m.begin();
            p != m.end(); p++)
  {
       // use *p or p->fld;
  }
  ```

- search

  ```
  auto p = m.find(elm);
  // returns m.end() or can
  // use *p or p->fld
  ```

but we said that

a blockchain is a ledger of transactions that is verifiable and permanent

verifiability problem #1: repudiation

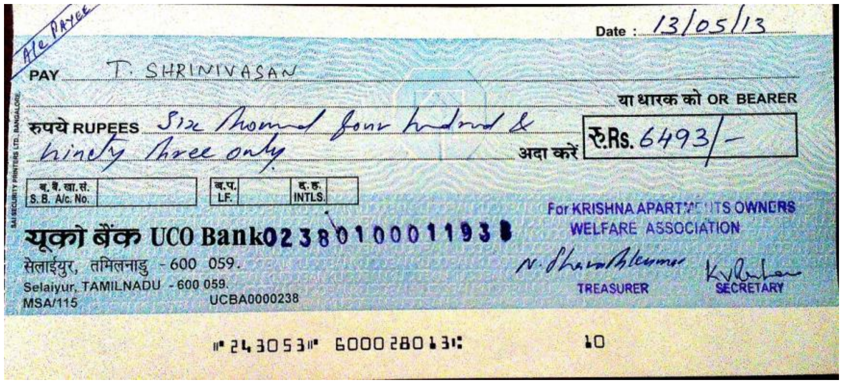
sender	receiver	amount
initial deposit	alice	100
initial deposit	bob	200
initial deposit	carol	300
initial deposit	dan	400
bob	alice	100
alice	carol	125
carol	**dan**	**100**
dan	alice	50
alice	bob	25
bob	dan	75

hey SBI, I never paid dan ฿100!

solution in today’s banks?

hey SBI, I never paid TS ₹6493!

yes, you did! here is your signature



A/c PAYEE

Date : 13/05/13

PAY T. SHRINIVASAN

या धारक को OR BEARER

रुपये RUPEES Six thousand four hundred & ninety three only

अदा करें

रु.Rs. 6493/-

ब. बै. खा. सं. S. B. A/c. No.		ब.प. LF.		ह. ह. INTLS.	

यूको बैंक UCO Bank 02380100011938

सेलाईयुर, तमिलनाडु - 600 059.
Selaiyur, TAMILNADU - 600 059.
MSA/115 UCBA0000238

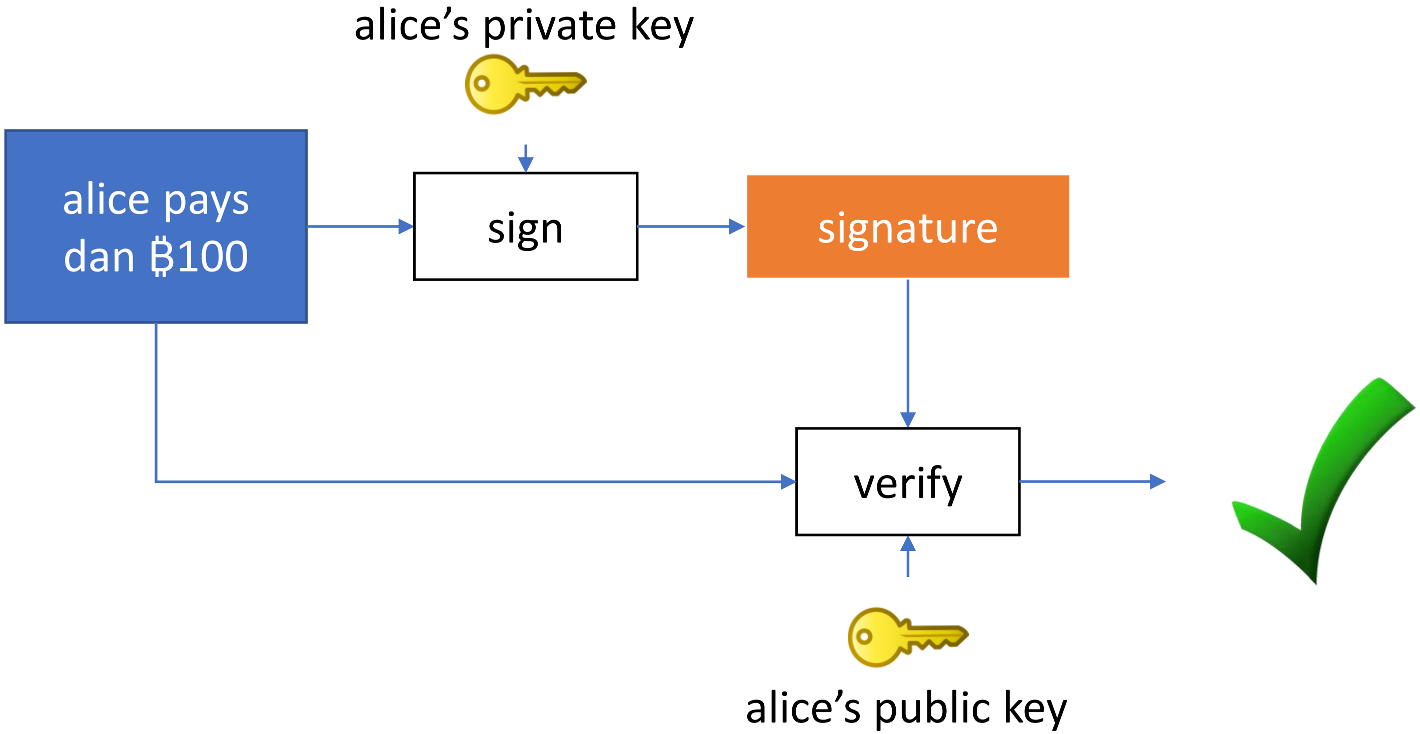
For KRISHNA APARTMENTS OWNERS WELFARE ASSOCIATION

TREASURER SECRETARY

SAI SECURITY PRINTERS LTD., BANGALORE.

⑈243053⑈ 600028013⑆ 10

in blockchains: digital signatures



alice's private key

alice pays dan ฿100

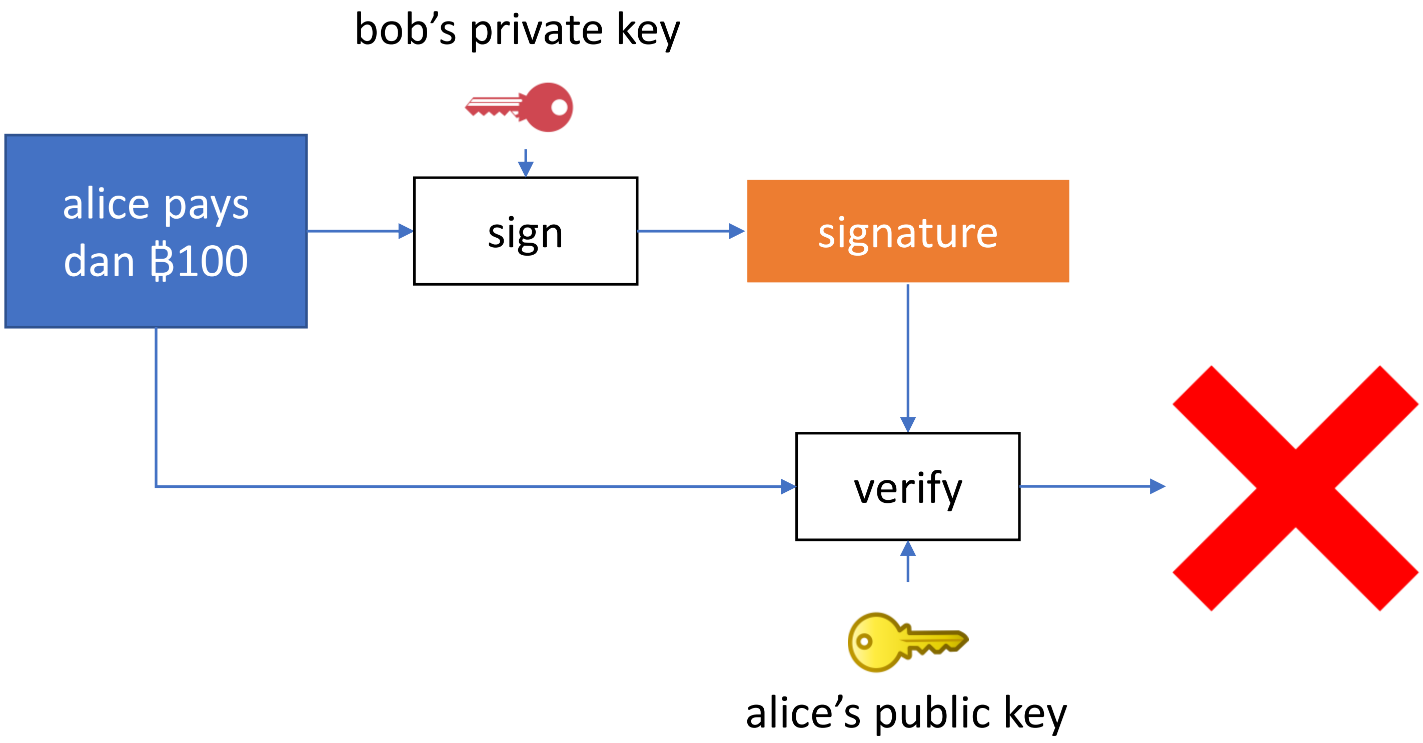
sign

signature

verify

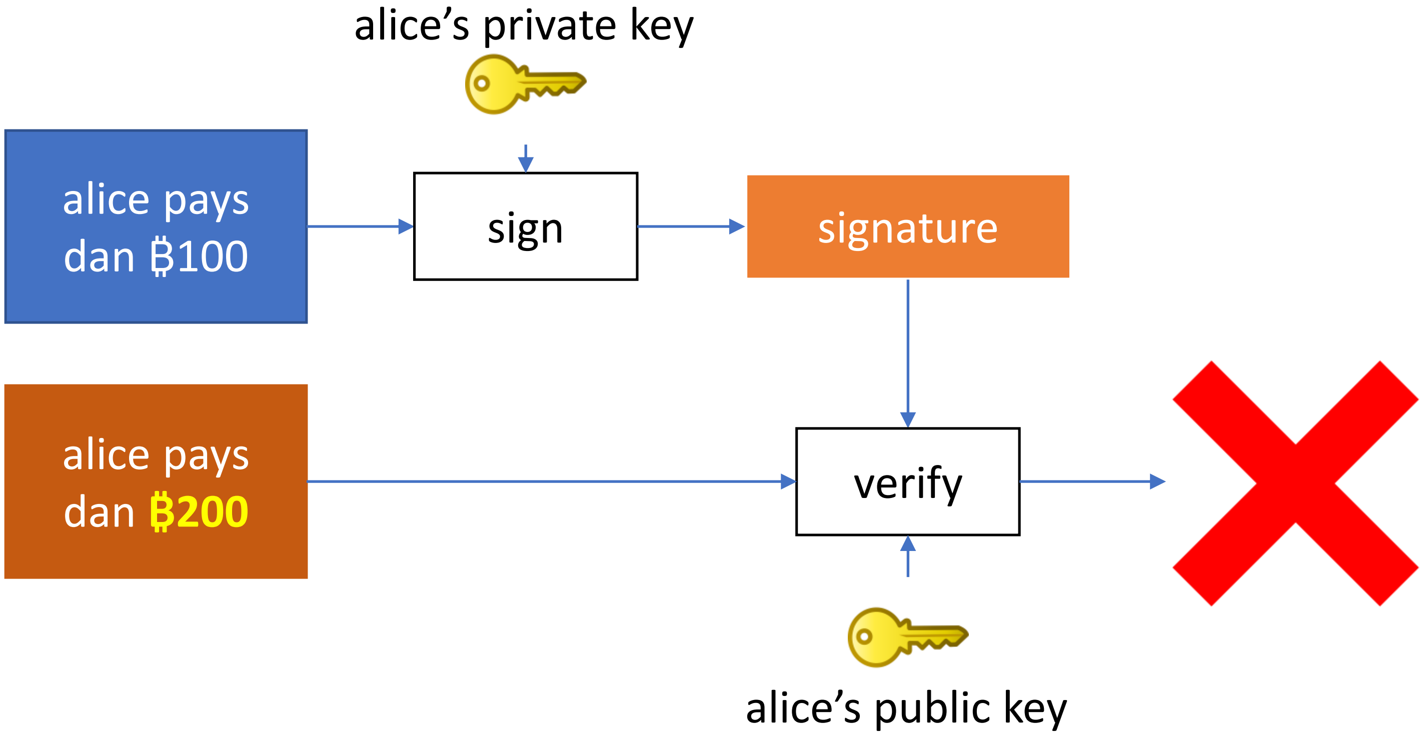
alice's public key

in blockchains: digital signatures



nobody else can forge signature without alice’s private key

in blockchains: digital signatures



verification will fail if the message is changed

a non-repudiable ledger

sender	receiver	amount	digital signature
initial deposit	alice	100	**0xFAD10A8DC**
initial deposit	bob	200	**0xBA2DC311C**
initial deposit	carol	300	**0x122938FAA1**
initial deposit	dan	400	**0x71123FFCB1**
bob	alice	100	**0x4801FFC3D1**
alice	carol	125	**0x8182830ABC**
carol	dan	100	**0xD1382C0124**
dan	alice	50	**0xFF14285714**
alice	bob	25	**0x91984B7521**
bob	dan	75	**0xBB0B304512**

definition: transaction validity (v2)

transaction is valid if

- sender’s balance is >= the amount being sent to receiver and
- and tx signature validation with sender’s public key succeeds

code for verifying signatures

```
rsa_public_key_t pubkey(
     DEREncodedKey.data(),
     DEREncodedKey.size());

bool success = pubKey.verify(
   (const uint8_t*) msg.data(), msg.size(),
   signature.data(), signature.size());
```

what’s the problem with the scheme?

where will the public keys come from?

what if bank maintains them?

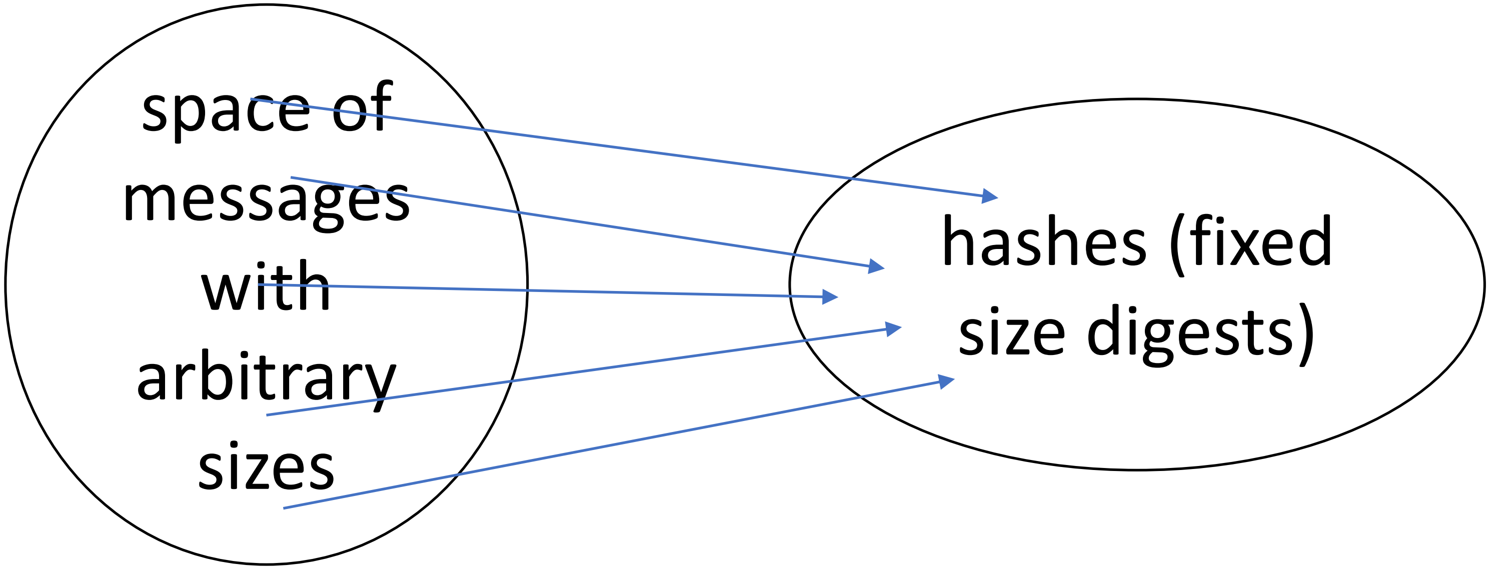
- bank will be able to forge sign (how?)

solution: tie a/c nos. and pubkeys

all account numbers = hash(public key)

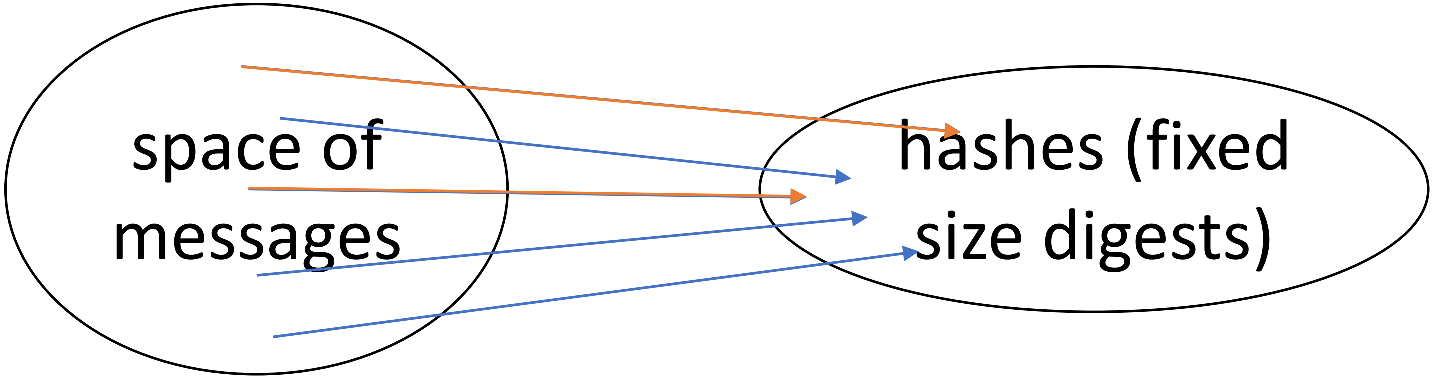
- customer chooses a/c no based on priv/pub keypair
- provides this account number to the bank
- transactions are signed using corresponding privkey

what is a cryptographic hash fn?


space of messages with arbitrary sizes
hashes (fixed size digests)

a mapping from arbitrary bytes to some fixed size (typically 16/32byte) digest

properties of crypto hash functions



pre-image resistance

- can’t find message given a hash

collision resistance

- if two messages are different, very likely hashes are different

blockchain transactions

send pubkey	send addr	recv addr	amt	digital signature
0xFFA1288…	0x18471C…	0x13831…	100	0xFAD10A8DC
…	…	…	…	…
…	…	…	…	…

a/c numbers are hashes of public keys

- from now on, we will call a/c nos. as addresses

public key is included in the transaction

- question: why not just use public keys as addresses?

problem: replay attacks

#	send pubkey	send addr	recv addr	amt	digital signature
1	0xFFA1288…	0x18471C…	0x13831…	100	0xFAD10A8DC
2	0x98B5B33..	0x13831…	0x32112…	50	0xD1ABC31A6
3	…	…	…	…	…
4	…	…	…	…	…

- after tx #1, 0x13831… has (at least) ฿100
- she spends some of this by giving 0x32112… ฿50

so what is the problem?

problem: replay attacks

#	send pubkey	send addr	recv addr	amt	digital signature
1	0xFFA1288…	0x18471C…	0x13831…	100	0xFAD10A8DC
2	0x98B5B33..	0x13831…	0x32112…	50	0xD1ABC31A6
3	0x98B5B33..	0x13831…	0x32112…	50	0xD1ABC31A6
4	…	…	…	…	…

- after tx #1, 0x13831… has (at least) ₿100
- she spends some of this by giving 0x32112… ₿50

so what is the problem?

- 0x32112 can replay the txn and get ₿50 again!

what’s the fix?

send pubkey	send addr	recv addr	change addr	amt	digital signature
0xFFA1288…	0x18471C…	0x13831…	0x4AC1..	100	0xFAD10A8..
0x98B5B33…	0x13831…	0x32112…	0xD1A2…	50	0x98B5B33..
…	…	…		…	…

- create a new address to send “change” (remaining balance) with each transaction
- after tx 2:

  0x13831 has ₿0; 0x32112 has ₿50; 0xD1A2 has ₿50

one last minor detail

send pubkey	send addr	recv addr	change addr	amt	tx hash	digital signature
0xFFA1288…	0x18471C…	0x13831…	0xB11A…	100	0x331A…	0xFAD10A8DC
…	…	…		…		…
…	…	…		…		…

- tx hash = hash(pubkey, send addr, recv addr, change addr, amt)
- tx sign = sign(tx hash, privkey)

the hash is needed for certain technical reasons in bitcoin

final transaction structure

```
class txn_t {
  vector<uint8_t> public_key;
  hash_result_t source_addr;
  hash_result_t dest_addr;
  hash_result_t change_addr;
  uint64_t amount;
  hash_result_t tx_hash;
  vector<uint8_t> tx_sign;
};
```

definition: transaction validity (v3)

send pubkey	send addr	recv addr	amt	tx hash	digital signature
0xFFA1288…	0x18471C …	0x13831…	100	0x331A…	0xFAD10A8DC
…	…	…	…		…
…	…	…	…		…

your first task is to implement these two checks in transaction.cpp

1. tx hash == hash(pubkey, send addr, recv addr, change addr, amt)
2. pubkey.verify(tx hash, tx sign) = 1
3. send addr (sender’s account) must have enough balance

code for computing hashes

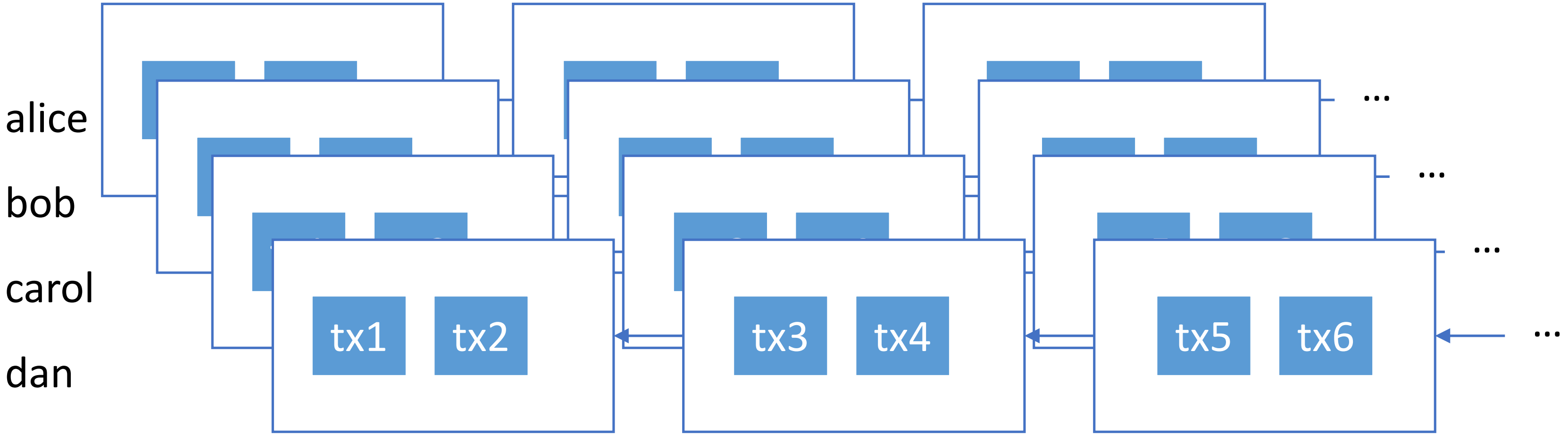
```
SHA256_CTX sha256;
SHA256_Init(&sha256);
SHA256_Update(&sha256, public_key.data(), public_key.size());
SHA256_Update(&sha256, source_addr.data(), source_addr.size());
SHA256_Update(&sha256, dest_addr.data(), dest_addr.size());
SHA256_Update(&sha256, change_addr.data(), change_addr.size());
SHA256_Update(&sha256, &amount, sizeof(amount));
SHA256_Final(tx_hash.data(), &sha256);
```

tx_hash = SHA256(public_key, source_addr, dest_addr, change_addr, amount)

back to the blockchain

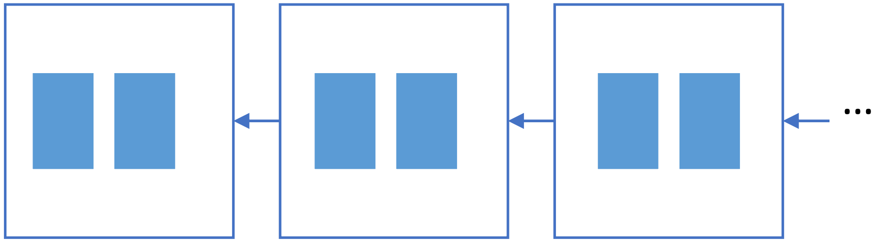
a blockchain is a ledger of transactions that is verifiable and permanent

permanence via public distribution



- all participants in the blockchain have a copy of it
- so every transaction is broadcast to everyone
- need a consensus algorithm to make sure everyone sees the same state when multiple people are using but we won’t go into this part

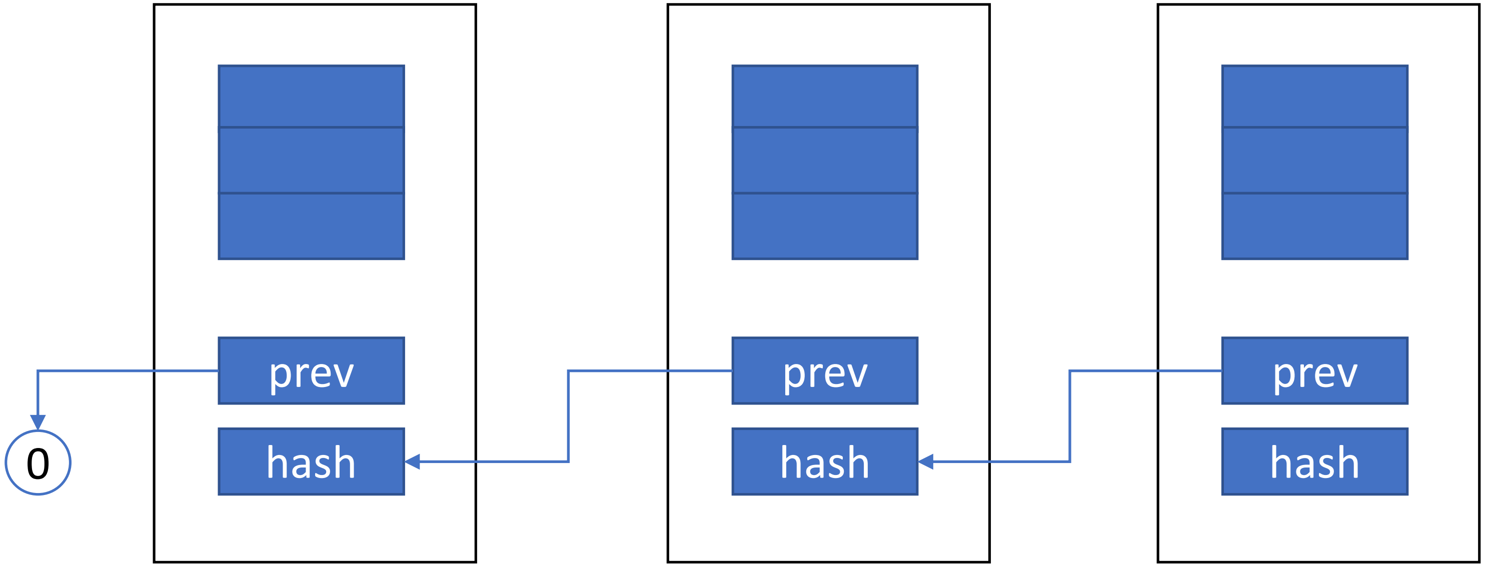
but now we have a problem



```
class block_t {
  vector<shared_ptr<txn_t> > txns;
  shared_ptr<block_t> prev_block;
};
```

how do we maintain `prev_block` pointers across different machines?

solution: hash pointers



- pointers refer to hashes of the data they point to
- not memory addresses

finally, where do bitcoins come from?

new bitcoins are created with each block

- reason has to do with the consensus operation

newly created bitcoins go to a “reward” address

- think of it as someone getting paid for taking the trouble of maintaining the ledger
- this payment is called a “block reward”

final block structure

```
class block_t {
  vector<shared_ptr<txn_t> > transactions;
  hash_result_t reward_addr;
  hash_result_t prev_hash;
  shared_ptr<block_t> prev_ptr; // why?
};
```

your tasks

- review code in cryptotest.cpp to see use of API functions
- implement txn_t::validate() [50 pts]
- (challenge) implement block_t::validate() [30 pts]
- (challenge) fix perf. bug in txn_t::update_balances [20]

git@git.cse.iitk.ac.in:spramod/bootcamp-day1.git

task #1: txn_t::validate()

write code for the following checks:

1. send addr = hash(pubkey)
2. tx hash == hash(pubkey, send addr, recv addr, change addr, amt)
3. pubkey.verify(tx hash, tx sign) = 1

task #2: block_t::validate()

write code for the following checks:

1. check each transactions `validate()` returns true
2. update the balances after each valid transaction
    - call `txn_t::update_balances` for this
3. check `blk_hash == hash(tx1->tx_hash, tx2->tx_hash, … , txn->tx_hash, reward_addr, prev_hash)`
4. Add the block reward to the balances

testing your code

- have given three tests cases: tests/t1.dat, t2.dat and t3.dat
- expected output is in tests/t1.txt, t2.txt and t3.txt
- will release three more after lunch
- scoring is based on correctness (correct output) and speed