Blockchain Technology And Applications

Sandeep K. Shukla

IIT Kanpur

C3I Center

Acknowledgement

- Much material in this course owe their ideas and existence to
  - Prof. Maurice Herlihy, Brown University
  - Prof. Hagit Attiya, Hebrew University
  - Prof. Arvind Narayanan, Princeton University
  - Prof. Joseph Bonneau, NYU
  - Prof. Pramod Subramanyan, IITK

What is Blockchain?

- A Linked List
  - Replicated
  - Distributed
  - Consistency maintained by Consensus
  - Cryptographically linked
  - Cryptographically assured integrity of data
- Used as
  - Immutable Ledger of events, transactions or time stamped data
  - Tamper resistant log
  - Platform to Create and Transact in Cryptocurrency
  - log of events/transactions unrelated to currency

Why a course on Blockchain?

- Have you seen the news lately?
  - Bitcoin
  - Ethereum
  - Blockchain for E-governance
  - Blockchain for supply chain management
  - Blockchain for energy management ……
  - Soon: Block chain for Nirvana
- Is it just a hype and hyperbole?
  - Hopefully this course will teach you otherwise
  - Even if you do not care about cryptocurrency and its market volatility

Let’s First talk about Banking (a la Arvind Narayanan)

How do you transact?

- Your write a check or do internet transaction to pay a payee
- Bank checks if you have balance > transaction amount
  - If yes, it debits your account by balance = balance - transaction_amount
    - credit's payee's account by payee.balance = payee.balance + transaction_amount
  - If no, the transaction is invalid and rejected.
- You can check your transaction list online, or check the monthly statement
- Who maintains the ledger?
  - Bank Does
  - What if Bank allows an invalid transaction go through
    - Invalid = you did not authenticate the transaction
    - Invalid = your balance was not sufficient but transaction was made

Bank Frauds

- You find a check was used to pay someone but you never wrote the check
  - Someone forged your check and/or signature
- You did sign a check for x amount, but the amount field was modified
  - How do you prove to the bank that an extra 0 was not there in your signing time?
- The monthly statement says that you did a transaction but you did not recall or the amount of a transaction is different from what you had done
  - Someone got your password, and possibly redirected OTP to another SIM (SIM Fraud)
  - Bank employees themselves might have done something
- How do you argue to the bank? (Non-repudiation)
- How do you argue that the amount was modified? (Integrity)
- Finally, do you tally your transactions when you receive your monthly statement?
  - Most people do not

Supply chain and provenance

- Your buy ice cream for your restaurant from supplier B
- Supplier B actually transports ice cream made in Company C's factory
- Upon delivery, you have been finding that your ice cream is already melted
- Who is responsible?
  - Supplier B is keeping it too long on the delivery truck?
  - Supplier B's storage facility has a temperature problem?
  - Supplier C says it's supplier B's fault as when picked up – ice cream was frozen
  - Supplier B says that when received, the temperature was too high, so C must have stored it or made it wrong
  - How do you find the truth?
    - Put temperature sensors in B's truck and storage, C's factory and storage, and sensor data is digitally signed by the entity where the sensor is placed and put in a log
    - You check the log – but B and C both have hacked the log and deleted some entries?
  - What to do?

Land Record

- Have you watched “Khosla ka Ghosla”?
- You buy a piece of land
- Someone else claims to own the land
- But the one who sold you the land showed you paper work
- Land registry office earlier said that the owner was rightful
- Now they say that they made a mistake – it was owned by the other person
- You already paid for the land – to the first person
- First person goes missing

  How does any one prove who changed the land record?
  - The government employees?

Then there is Aadhaar

- E-KYC Logs
- Shown to you by UIDAI
- How do you know they did not delete important log events?
- Do you Trust UIDAI?

A student Online Grade Submission and Management System

Professor	Course	Grade
1	ESC101	D
2	CS698	D
3	CS425	D
4	CS771	D

This course is not about bitcoin or currency: Why?

Why not bitcoin? (2)

HOME ABOUT OUR BLOG | BLOCKONOMICS WEBSITE

Frederick Coleman Follow
Manager Media and Communications— Blockonomics
Jun 16, 2017 · 6 min read

The Dark Side of Bitcoin: Illegal Activities, Fraud, and Bitcoin

Why not bitcoin? (3)

Why not bitcoin? (4)

BLOCKCHAIN VC TOP 10 STORIES | SXSW HACKATHON

Proof of work, or proof of waste?

Bitcoin and the energy usage dilemma

If you're moderately well versed in the blockchain ecosystem, then you would have by now come across the energy consumption debate that is taking place in relation to Bitcoin Mining. Yes, Bitcoin Mining requires a lot of energy, and not a modest amount either. Mining (under the current protocol of Bitcoin's *proof of work* algorithm) is **deliberately** consumptive of energy.

Why not bitcoin? (5)

Why not bitcoin? (6)

Why no money business?

RIBE | ABOUT | RSS

cyberscoop

BROUGHT TO YOU BY SNG

NMENT | TRANSPORTATION | HEALTHCARE | TECHNOLOGY | FINANCIAL | WATCH | LISTEN | ATTEND | COMMUN

TECHNOLOGY

The curious case of the missing Mt. Gox bitcoin fortune

Quadriga Exchange – Loss of 145 USD worth cryptocurrency

A crypto exchange may have lost $145 million after its CEO suddenly died

By Daniel Shane, CNN Business

Updated 0251 GMT (1051 HKT) February 6, 2019

TOP STORIES

Steve Harvey ha
Miss Universe m

A wildlife conser
mauled by her o

Recomm

Why no money business? (2)

Bitcoins and other cryptocurrencies

- Too much interest by investors to park their assets
- Less use as a medium of value exchange
- Private Key stealing or private keys at exchanges — risk
- Coding vulnerabilities — risk
- Volatility
- Energy Waste — climate impact
- Too much concentration in one country — risk
- Regulatory risk
- Usage for criminal activities — Silk Road

Again, What is a blockchain?

- Blockchain technology is a digital innovation that has the potential to significantly impact trusted computing activities and therefore cybersecurity concerns as a whole.
- Attractive properties of Blockchain
  - Log of data with digital signature
  - Immutable (once written – cryptographically hard to remove from the log)
  - Cryptographically secure – privacy preserving
  - Provides a basis for trusted computing on top of which applications can be built

Trust Model

- Cyber Security is all about who you trust?
  - Trust your hardware to not leak your cryptographic keys?
  - Trust your O/S to not peek into your computation memory?
  - Trust your hypervisor to not mess up your process memory?
  - Trust your application to not be control hijacked or attack other applications?
- Where is your trust anchor?
  - Hardware?
  - Operating system?
  - Application?
  - Manufacturer?

Trust Model (2)

- In many real life transactional activities – trust model is the inverse of the threat model
  - Do you trust your bank to not take out small amounts from your balance all the time? (Watch – “Office Space”)
  - Do you trust the department of land records to keep your record’s integrity?
  - Do you trust UIDAI officials to keep your aadhaar data from unauthorized access?
  - Do you trust your local system admins to not go around your back and change settings, leak passwords, change database entries, and remove their action from system logs?
  - In the patch management system of your enterprise, are the patches being put -- all have digital certificates? Who put them? Do you trust your employees to do the correct thing and not put a malware as patch?

Digital Currency ideas

Arvind Narayanan's CryptoCoin-Version 1

CryptoCoin-v.1

Can you create your own digital coin?

How about ?

signed by pk_{you}
CreateCoin [uniqueCoinID]

pk_{you} -- your private key

puk_{you} – your public key

Give your friend a coin you created

Your friend can pass the coin to his friend Arvind

Arvind owns it now

double-spending attack

Who owns the coin now?

Arvind or Ravi?

Major challenge in Digital Currency Design

How do you stop double-spending?

Arvind Narayanan's CryptoCoin-version 2

Double Spending Proof Digital Currency

CryptoCoin-v.2

Transaction History Tracking

optimization: put multiple transactions in the same block

CreateCoins transaction creates new coins

transID: 73	type:CreateCoins	
coins created		
num	*value*	*recipient*
0	3.2	0x...
1	1.4	0x...
2	7.1	0x...

coinID 73(0)

coinID 73(1)

coinID 73(2)

Who keeps track the transaction History?

Who Arbitrates when a double spending is tried?

Which of the transactions is valid?

PayCoins transaction consumes (and destroys) some coins,
and creates new coins of the same total value

consumed coinIDs: 68(1), 42(0), 72(3)		
coins created		
num	*value*	*recipient*
0	3.2	0x...
1	1.4	0x...
2	7.1	0x...
signatures		

Valid if:
-- consumed coins valid,
-- not already consumed,
-- total value out = total value in, and
-- signed by owners of all consumed coins

Immutable coins

Coins can’t be transferred, subdivided, or combined.

But: you can get the same effect as subdivision by using transactions

- create new transactions
- consume your coin
- pay out two new coins to yourself

Trusted Third Party

Crucial question:

You become the central Trusted Party keeping track of transaction history, arbitrarting validity of transactions

Why should people trust you??

Back to the supply Chain Story (Herlihy)

Ice Cream Supply Chain to Restaurant A from Factory C via Supplier B

Ice cream is melted

Ice Cream Supply Chain to Restaurant A from Factory C via Supplier B

Use IoT to create non-repudiation

IoT sensors sends real-time data to the server at Restaurant A to periodically show the factory and the truck temperatures to Restaurant A

What can go wrong?

- IoT sensor data may be intercepted by a middle man and changed before it reaches the server (**data integrity**)
- IoT sensors may be stopped and old readings may be replayed (**replay attack**)
- What the server gets purportedly from factory C, may be manufactured by supplier B (**Authenticity**)
- If restaurant A claims that C's temperature reading shows that ice cream was melting in the storage, C can say that message you received is not from me – there was an MITM attack (**repudiation**)

- So restaurant A will not be able to pinpoint any one in the supply chain with full confidence!!

What can be done?

- Use a message integrity proof (Hashing)
- Use digital signature of the individual IoT devices (Authenticity and non-repudiation)
  - assuming the digital signatures cannot be forged
  - private keys are kept safe
- Use authentic time stamping with the IoT data before hashing for integrity (avoid replay attacks)
- So now factory A can pinpoint with some basic security assumptions about this infrastructure

Concurrency Issue

- A has other suppliers for other goods required for its business (multiple concurrent supply chains)
- B and C has multiple other consumers of their services
- So if there are N suppliers who are also consumers of some of these entities, we have an N^2 messaging problem

A offers that every one can look up their data from my server, so you can get linear number of messaging

But do you trust A as purveyors of your data?

Solutions?

- Have a trusted authority or a cloud provider to become a publish-subscribe service provider
- Every supplier sends their IoT data with message integrity, authentication code etc., to the cloud server
  - Every consumer subscribes to the events they are interested in on the cloud
  - Every supplier becomes authenticated data generator on the cloud

What if the cloud provider cannot be trusted?

Create a framework on which data is crowd sourced, validated by the crowd for the crowd?

- You get a block chain
- But now the question is as concurrent messages come in to this framework, how do you order them?

DISTRIBUTED CONCENSUS IS REQUIRED TO DECIDE

1. of all messages coming in concurrently how are they ordered

2. But if some of the crowd are malicious, and tries to allow data that are wrong, or ordered wrong?

3. You need Byzantine fault-tolerant consensus

Conclusion of the First Lecture

- Blockchain is about
  - Distributed Record Keeping
  - Trust Model varies – but usually single point of trust is not good
  - Based on Trust Model –
    - Permissioned Blockchain
    - Non-permissioned or public block chain
    - Also, private blockchain
  - Data integrity (No one has tampered with the data after its creation)
  - Authenticated Transactions or event logging
  - Strong Cryptographic Application
- Blockchain is certainly not ONLY
  - Cryptocurrency
  - In this course, cryptocurrency will be avoided

Summary of Lecture 1

- What did you learn today?
  - The need to learn about block chain technology and its applications
  - Bitcoin and Cryptocurrencies are only an example application of the technology
  - This course is not about Bitcoin or Cryptocurrency – but more on the technology and applications
  - Trust Model determines whether you need blockchain and if so – what kind – permissioned/permissionless/private
  - Basic issues leading to the Bitcoin (Trust model again)
  - Basic issues in supply chain provenance and integrity and trust model
  - Concurrency is important to take into account
  - Fault-tolerant Consensus is a requirement for handling concurrency and trust model issues

Blockchain Technology And Applications

Sandeep K. Shukla

IIT Kanpur

C3I Center

Acknowledgement

- Much material in this course owe their ideas and existence to
- Prof. Maurice Herlihy, Brown University
- Prof. Hagit Attiya, Hebrew University
- **Prof. Arvind Narayanan, Princeton University**
- **Prof. Joseph Bonneau, NYU**
- Dan Boneh (Stanford University)
- John C. Mitchell (Stanford University)
- Nicolai Zeldovich (MIT)
- Jungmin Park (Virginia Tech)
- Patrick Schaumont (Virginia Tech)
- C. Edward Chow
- Arun Hodigere
- Mike Freedman, Princeton University

Today is all about essential Cryptography for Blockchain

- crypto basics that are essential for blockchain technology
- Hash functions and their properties
- Public Key Cryptosystems
- Digital Signatures
- Hash Puzzles
- Hash Pointers
- Merkle Data Structures

Basics concepts of Cryptography

Cryptography: Basic Terminology

- Plaintext (or cleartext)
  - The message.
- Encryption (encipher)
  - Encoding of message.
- Ciphertext
  - Encrypted message.
- Decryption (decipher)
  - decoding of ciphertext

Encryption and Decryption

The following identity must hold true:

$$D(C) = M, \text{ where } C = E(M)$$

$$M = D(E(M))$$

Cryptography: Algorithms and Keys

- A method of encryption and decryption is called a **cipher.**
- Generally there are two related functions: one for encryption and other for decryption.
- Some cryptographic methods rely on the secrecy of the algorithms.
  - Such methods are mostly of historical interest these days.
- All modern algorithms use a **key** to control encryption and decryption.
- Encryption key may be different from decryption key.

Key Based Encryption/Decryption

<u>Symmetric Case:</u> both keys are the same or derivable from each other. $K_1 = K_2$.

<u>Asymmetric Case:</u> keys are different and not derivable from each other. $K_1 \mathrel{!=} K_2$

Secrete Key Cryptography

K is the secret key shared by both the sender (S) and receiver (R).

Secret Key Cryptography

- Also called symmetric or single-key algorithms.
- The encryption and the decryption key are the same.
- Techniques based on a combination of substitution and permutation.
- Stream ciphers: operate on single bit or byte.
- Block ciphers: operate on blocks (typically 64/128/256… bits)
- Advantage: simple, fast.
- Disadvantage: ***key exchange, key management***.
- Examples: DES,RC4, IDEA, Blowfish, AES, etc.

Private Key Cryptosystem (Symmetric)

Symmetric Key - Issues

Key management, keys required = (p*(p-1))/2 or:

Secrete Key Assurances

- Confidentiality
  - is assurance that only owners of a shared secrete key can decrypt a message that has been encrypted with the shared secrete key
- Authentication
  - is assurance of the identify of the person at the other end of the line (use challenge and response protocols)
- Integrity
  - is assurance that a message has not been changed during transit and is also called message authentication (use message fingerprint)
- Non-repudiation
  - is assurance that the sender cannot deny a file was sent. This cannot be done with secrete key alone (need trusted third party or public key technology)

Example: non-repudiation

- Scenario 1:
  - Alice sends a stock buy request to Bob
  - Bob does not buy and claims that he never received the request
- Scenario 2:
  - Alice sends a stock buy request to Bob
  - Bob sends back an acknowledge message
  - Again, Bob does not buy and claims that he never received it
  - Alice presents the ack message as proof
- Can she prove that the ack message was created by him?

DES (Data Encryption Standard)

- In 1972, NIST (National Institute of Standards and Technology) decides to assist the development of a secure cryptographic method.
- In 1974, it settled on DES, which was submitted by IBM and is the Data Encryption Algorithm developed by Horst Feistel.
- NSA shortened the secrete key to 56 bits from 128 bits originally proposed by IBM.
- Initially intended for 10 years. DES reviewed in 1983, 1987, 1993.
- In 1997, NIST solicited candidates for a new secrete key encryption standard, Advanced Encryption Standard (AES).
- In Oct 2000, NIST selected Rijndael. (www.nist.gov/AES)

Cycling through DES keys

- In 1977, a 56-bit key was considered good enough.
  - Takes 1,000 years to try all keys with 56 1's and 0's at one million keys per second
- In Jan 1997, RSA Data Security Inc. issued "DES challenge"
  - DES cracked in 96 days
  - In Feb 1998, distributed.net cracked DES in 41 days
  - In July 1998, the Electroic Frontier Foundation (EFF) and distributed.net cracked in 56 hours using a $250K machine
  - In Jan 1999, the team did in less than 24 hours
- Double and Triple DES
  - Double DES only gives 2**57 = 2 x 2**56, instead of 2**112, due to *meet-in-the-middle* attack.
  - Triple DES recommended, but managing three keys more difficult

Public Key Cryptography

$K_{R(pub)}$ is Receiver's public key and $K_{R(pri)}$ is Receiver's private key.

Establishing Shared Secrete

This Photo by Unknown Author is licensed under CC BY-SA

Problem Statement

- Suppose Alice has an channel for communicating with Bob.
- Alice and Bob wish to use this channel to establish a shared secret.
- However, Eve is able to learn everything sent over the channel.
- If Alice and Bob have no other channel to use, can they establish a shared secret that Eve does not know?

Public Key Cryptographic Algorithms

Find a hard math problem, that is easy to compute in the forward direction, but is difficult to solve in the reverse direction, unless you have some special knowledge.

Public Key Cryptosystem

This Photo by Unknown Author is licensed under CC BY-SA

General Strategy

- A public key is used to encrypt a message that can be decrypted only by the matching private key.
- Bob can use Alice's public key to encrypt messages. Only Alice can decrypt the message.
- Similarly, Alice can also use Bob's public key.
- Alice and Bob exchange information, each keeping a secret to themselves.
- The secrets that they keep allow them to compute a shared secret.
- Since Eve lacks either of these secrets she is unable to compute the shared secret.

Asymmetric Algorithms

- Also called public-key algorithms.
- Encryption key is different from decryption key.
- Furthermore, one cannot be calculated from other.
- Encryption key is often called the **public key** and decryption key is often called the **private key**.
- Advantages: better key management.
- Disadvantages: slower, more complex.
- Both techniques are complementary.
- Examples: RSA, Diffie-Hellman, El Gamal, etc.

Cryptographic Hash Functions

- Hash function:
  - takes an arbitrary length string as input
  - produces a fixed-size output (e.g 256 bits)
    - Easy to compute
    - Almost impossible to reverse
  - Security properties:
    - collision-resistant
    - Hides the original String
      - Almost impossible to get the original string from the output
    - puzzle-friendly

Hash property 1: Collision-resistance

It is computationally NOT feasible to find x and y such that

x != y and H(x)=H(y)

However, for a weak hash function collisions may be feasible to find:

Examples of hash functions for which collisions are found feasibly:

MD-5, SHA-1

Brute-forcing collision

try 2^{130} randomly chosen inputs

99.8% chance that two of them will collide

This works no matter what H is …

… but it takes too long to matter

Even if each input checking takes 100 ms, we are talking about $2^{130} \times 0.1\ seconds$ which is $\frac{1}{5} \times 2^{129}\ seconds = 4.3\ \times\ 10^{30}\ years$

Are there any hash functions for which efficient collision finding algorithms exist?

- For some H
- But for others none known yet (SHA-256, SHA3 etc)

No H has been proven collision-free.

Many H has been proven to have collisions.

Examples: SHA-1

https://security.googleblog.com/2017/02/announcing-first-sha1-collision.html

Application: Hash as message digest

- If we know H(x) = H(y),
  - it's safe to assume that x = y.
- To recognize a file that we saw before,
  - just remember its hash.
- Useful because the hash is small.

Hash property 2: Hiding

We want something like this:

Given H(x), it is infeasible to find x.

Hash property 2: Hiding

- Hiding property:
  - If r is chosen from a probability distribution that has *high min-entropy*, then given H(r | x), it is infeasible to find x.

- High min-entropy means
  - the distribution is "very spread out", so that no particular value is chosen with more than negligible probability.

To know a bit more on this: https://crypto.stackexchange.com/questions/66097/why-is-min-entropy-significant-in-cryptography

Application: Commitment

Want to “seal a value in an envelope”, and
“open the envelope” later.

Commit to a value, reveal it later.

Commitment API

(*com, key*) := commit(*msg*)
match := verify(*com, key, msg*)

To seal *msg* in envelope:
  (*com, key*) := commit(*msg*) -- then publish *com*
To open envelope:
  publish *key, msg*
  anyone can use verify() to check validity

Commitment API

(*com, key*) := commit(*msg*)
match := verify(*com, key, msg*)

Security properties:
  Hiding: Given *com*, infeasible to find *msg*.
  Binding: Infeasible to find *msg != msg'* such that
    verify(commit(*msg*), *msg'*) == true

Commitment API

commit(*msg)* := (H(*key | msg*), H(*key)*)
where *key* is a random 256-bit value
verify(*com, key, msg*) := (H(*key | msg) == com*)

Security properties:
Hiding: Given H(*key | msg)*, infeasible to find *msg*.
Binding: Infeasible to find *msg != msg'* such that
H(*key | msg) ==* H(*key | msg')*

Hash property 3: Puzzle-friendly

Puzzle-friendly:
For every possible output value y,
if k is chosen from a distribution with high min-entropy,
then it is infeasible to find x such that $H(k \mid x) = y$.

Application: Search puzzle

Given a "puzzle ID" *id* (from high min-entropy distrib.),
and a target set *Y*:
Try to find a "solution" *x* such that
$H(id \mid x) \in Y$.

Puzzle-friendly property implies that no solving strategy is much better than trying random values of *x (Brute-force)*

SHA-256 hash function

Theorem: If c is collision-free, then SHA-256 is collision-free.

SHA-256 Padding

- To ensure that the message has length multiple of 512 bits:
  - first, a bit 1 is appended
  - next, k bits 0 are appended, with k being the smallest positive integer such that $l + 1 + k = 448 \, mod \, 512$
    - where l is the length in bits of the initial message
- finally, the length $l < 264$ of the initial message is represented with exactly 64 bits, and these bits are added at the end of the message.
- The message shall always be padded, even if the initial length is already a multiple of 512

Hash Pointers and Data Structures

Hash Pointers

- hash pointer is:
  - pointer to where some info is stored, and
  - (cryptographic) hash of the info

- if we have a hash pointer, we can
  - ask to get the info back (locate)
  - verify that it hasn't changed (integrity)

Pictorial Representation of Hash Pointers

Represent hash pointers like this

key idea:

build data structures with hash pointers

linked list with hash pointers = “block chain”

use case: tamper-evident log

Detecting tampering

use case: tamper-evident log

Binary tree with hash pointers = “Merkle tree”

Proving membership in a Merkle tree

show O(log n) items

Advantages of Merkle trees

- Tree holds many items
  - but just need to remember the root hash
- Can verify membership in O(log n) time/space

- Variant: sorted Merkle tree
- can verify non-membership in O(log n)
- (show items before, after the missing one)

More generally ...

can use hash pointers in any pointer-based data structure that has no cycles

Digital Signatures

What we want from signatures

Only you can sign, but anyone can verify

Signature is tied to a particular document

can’t be cut-and-pasted to another doc

API for digital signatures

(sk, pk) := generateKeys(keysize)

sk: secret signing key

pk: public verification key

sig := sign(sk, message)

isValid := verify(pk, message, sig)

can be randomized algorithms

Requirements for signatures

"valid signatures verify"

verify(pk, message, sign(sk, message)) == true

"can't forge signatures"

adversary who:

knows pk

gets to see signatures on messages of his choice

can't produce a verifiable signature on another message

Practical stuff...

- algorithms are randomized
  - need good source of randomness
- limit on message size
  - fix: use Hash(message) rather than message
- trick: sign a hash pointer
  - signature “covers” the whole structure

- Bitcoin uses ECDSA standard
  Elliptic Curve Digital Signature Algorithm

- Relies on hairy math
  will skip the details here

https://www.maximintegrated.com/en/design/technical-documents/tutorials/5/5767.html

- Good randomness is essential
  If the randomness is not good in generateKeys() or sign() ?
  probably will leak your private key

Public Keys as Identities

Useful trick: public key == an identity

if you see *sig* such that *verify(pk, msg, sig)==true*,
think of it as

pk says, "*[msg]*".

to "speak for" *pk*, you must know matching secret key *sk*

How to make a new identity

create a new, random key-pair *(sk, pk)*

pk is the public "name" you can use

[usually better to use Hash(pk)]

sk lets you "speak for" the identity

you control the identity, because only you know *sk*

if *pk* "looks random", nobody needs to know who you are

Decentralized identity management

- Anybody can make a new identity at any time
  make as many as you want!

- No central point of coordination

These identities are called “addresses” in Bitcoin.

Privacy

Addresses not necessarily directly connected to real-world identity.

But observer can link together an address's activity over time, make inferences (Pseudo Anonymity)

Summary for Lecture 2

- Very basic concepts of cryptography that are used in blockchain
  - Secure Hash Functions
    - Collision Resistance
    - Hiding Property
    - Puzzle Friendliness
  - Very basic idea about SHA-256 which is used in bitcoin Blockchain
  - Digital Signatures
    - Public, Private key pair
    - Signature for authentication and non-repudiation
  - Hash Pointers
    - Linking blocks with hash pointers
    - Merkle tree data structure (Hash pointer based tree)
  - Public Key as identity

Learning Done

- We will not go into further depth of cryptography
  - You should look up more on RSA to know how they generate public and private key
  - Look up Diffie-Hellman Cryptosystem basics
- If interested read more about
  - SHA-256, SHA-512, SHA-3
  - Look up collision in MD5, SHA-1
  - Elliptic Curve crypto system
- We learned how to conceptualize hash pointers
  - As you will see in Hash Pointers, you necessarily need the hash but not always the pointer
  - How to build Tree structures with Hash Pointers (Merkle Data structure)
  - Look up more on KSI (Keyless Signature Infrastructure) of Estonia
    - How they exploit Merkle tree

Next Lecture

- A real simple blockchain example and exercise
  - Due to Prof. Pramod Subramanyan (IITK)

Blockchain Technology And Applications

Sandeep K. Shukla

IIT Kanpur

C3I Center

Acknowledgement

- The material of this lecture and the corresponding exercise was created by Prof. Pramod Subramanyan (IITK)

how to:
create your own blockchain

By

Prof. Pramod Subramanyan

in the rest of this lecture

we will learn how to

1. build a ledger of transactions
2. Make it verifiable and permanent
3. And explain your first programming assignment (not for grading but your self learning)

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 something 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 something with *p;
}
```

- iteration (v2)

```
for (auto elem : vec) {
    // do something 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

solution in today's banks?

hey SBI, I never paid TS ₹6493!

yes, you did! here is your signature

in blockchains: digital signatures

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?

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()
- (challenge) implement block_t::validate()
- (challenge) fix perf. bug in txn_t::update_balances

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

Outline of blockchain.h

```
class block_t {
private:
    bool valid;
    balance_map_t balances;

public:
    unsigned length;
    block_t();
    block_t(std::shared_ptr<block_t> prev_block);
    hash_result_t reward_addr;
    std::vector< std::shared_ptr<txn_t> > transactions;
     hash_result_t prev_hash;
     hash_result_t blk_hash;
     std::shared_ptr<block_t> prev_block; ….
```

Outline of transactions.h

```
typedef std::map<hash_result_t, uint64_t> balance_map_t;

struct txn_t {
    uint8_vector_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;
    uint8_vector_t tx_sign;
    bool valid;
……
```

Outline of crypto.h

```
typedef std::vector<uint8_t> uint8_vector_t;

class hash_result_t {
        uint8_t bytes[SHA256_DIGEST_LENGTH];
public:
        static const int SIZE;
        hash_result_t();
        hash_result_t(uint8_t bytesIn[SHA256_DIGEST_LENGTH]);
        hash_result_t(const hash_result_t& other);
        void set_hash_from_data(const uint8_vector_t& data);
        void set_hash_from_data(const uint8_t* data, size_t sz);
        hash_result_t& operator=(const hash_result_t& other) {
                std::copy(other.bytes, other.bytes + size(), bytes);
                return *this;
                }
        bool operator==(const hash_result_t& other) const;
        bool operator!=(const hash_result_t& other) const {   return !(*this == other); }
        bool operator<(const hash_result_t& other) const;
        uint8_t& operator[](unsigned i);
```

Creating coinbase Block & Regular Block

```
block_t::block_t()
    : valid(false)
    , length(1)
    , prev_block(NULL)
{
    reset_balances();
}
```

```
block_t::block_t(std::shared_ptr<block_t> prev)
    : valid(false)
    , length(prev->length + 1)
    , prev_hash(prev->blk_hash)
    , prev_block(prev)
{
    reset_balances();
}
```

Steps you need to do

- Download Ubuntu 18.04 from ubuntu site if you are already not using ubuntu
- If you do not want your machine to be dual-booted, install Vbox from oracle and use virtual machine to run Ubuntu
- When you get into ubuntu, please clone openressl library from github https://github.com/libressl-portable/portable
- Use sudo apt-get install for the followings libboost-dev, automake, autoconf, git, libtool, perl, g++
- After you have installed the above, you have to follow the instructions in README.md that is in the directory portable-manager to build and install this library
- This is the SSL and Crypto library that you will need to do all the crypto functions required

Steps (2)

- Then download the code for blockchain exercise – and follow the README.md file within the src directory to build it
- You will need to then write the functions required for the homework.

Blockchain Technology And Applications

Sandeep K. Shukla

IIT Kanpur

C3I Center

Acknowledgement

- The material of this lecture is mostly due to Prof. Arvind Narayanan's Lecture at Princeton and his book on Bitcoin (Chapter 3)

Mechanics of Bitcoin

Recap: Bitcoin consensus

Bitcoin consensus enables us to have:

- An Append-only ledger of transactions
- Decentralized consensus with probabilistic guarantee
- Miners incentivized to validate transactions

To incentivize miners within the blockchain we need the blockchain to have a currency – blockchains that do not generate currency cannot have miners

Bitcoin transactions

An account-based ledger (*not* Bitcoin)

time

Create 25 coins and credit to Alice$_{\text{ASSERTED BY MINERS}}$
Transfer 17 coins from Alice to Bob$_{\text{SIGNED(Alice)}}$
Transfer 8 coins from Bob to Carol$_{\text{SIGNED(Bob)}}$
Transfer 5 coins from Carol to Alice$_{\text{SIGNED(Carol)}}$
Transfer 15 coins from Alice to David$_{\text{SIGNED(Alice)}}$

is this valid?

might need to scan backwards until genesis!

SIMPLIFICATION: only one transaction per block

A transaction-based ledger (Bitcoin)

Merging value

SIMPLIFICATION: only one transaction per block

Joint payments

SIMPLIFICATION: only one transaction per block

The real deal: a Bitcoin transaction

metadata

input(s)

output(s)


```
{
    "hash":"5a42590fbe0a90ee8e8747244d6c84f0db1a3a24e8f1b95b10c9e050990b8b6b",
    "ver":1,
    "vin_sz":2,
    "vout_sz":1,
    "lock_time":0,
    "size":404,
    "in":[
     {
      "prev_out":{
       "hash":"3be4ac9728a0823cf5e2deb2e86fc0bd2aa503a91d307b42ba76117d79280260",
       "n":0
      },
        "scriptSig":"30440..."
     },
     {
      "prev_out":{
       "hash":"7508e6ab259b4df0fd5147bab0c949d81473db4518f81afc5c3f52f91ff6b34e",
       "n":0
      },
      "scriptSig":"3f3a4ce81...."
     }
    ],
    "out":[
     {
      "value":"10.12287097",
      "scriptPubKey":"OP_DUP OP_HASH160 69e02e18b5705a05dd6b28ed517716c894b3d42e OP_EQUALVERIFY
OP_CHECKSIG"
     }
    ]
}
```

The real deal: transaction metadata

transaction hash

housekeeping

“not valid before”

housekeeping


```
{
    "hash":"5a42590...b8b6b",
     "ver":1,
     "vin_sz":2,
     "vout_sz":1,
     "lock_time":0,
     "size":404,
...
}
```

The real deal: transaction inputs

Previous transaction

signature

(more inputs)

```
"in":[
    {
      "prev_out":{
        "hash":"3be4...80260",
        "n":0
      },
    "scriptSig":"30440....3f3a4ce81"
    },
   ...
],
```

The real deal: transaction outputs

output value

recipient address??

(more outputs)


```
"out":[
  {
   "value":"10.12287097",
   "scriptPubKey":"OP_DUP OP_HASH160 69e...3d42e OP_EQUALVERIFY OP_CHECKSIG"
  },
  ...
 ]
```

Bitcoin scripts

Output “addresses” are really *scripts*

```
OP_DUP
OP_HASH160
69e02e18...
OP_EQUALVERIFY OP_CHECKSIG
```

Input “addresses” are *also* scripts

scriptSig

30440220...
0467d2c9...

scriptPubKey

OP_DUP
OP_HASH160
69e02e18...
OP_EQUALVERIFY OP_CHECKSIG

TO VERIFY: Concatenated script must execute completely with no errors

Bitcoin scripting language ("Script")

Design goals

- Built for Bitcoin (inspired by Forth)
- Simple, compact
- Support for cryptography
- Stack-based
- Limits on time/memory
- No looping

image via Jessie St. Amand

Bitcoin script execution example

Bitcoin script instructions

256 opcodes total (15 disabled, 75 reserved)

- Arithmetic
- If/then
- Logic/data handling
- Crypto!
  - Hashes
  - Signature verification
  - Multi-signature verification

OP_CHECKMULTISIG

- Built-in support for joint signatures
- Specify *n* public keys
- Specify *t*
- Verification requires *t* signatures

BUG ALERT: Extra data value popped from the stack and ignored

Bitcoin scripts in practice (as of 2014)

- Most nodes whitelist known scripts
- 99.9% are simple signature checks
- ~0.01% are MULTISIG
- ~0.01% are Pay-to-Script-Hash
- Remainder are errors, proof-of-burn

Proof-of-burn

nothing’s going to redeem that ☹

OP_RETURN
<arbitrary data>

Should senders specify scripts?

Idea: use the hash of redemption script

Pay to script hash

Applications of Bitcoin scripts

Example 1: Escrow transactions

Example 2: Green addresses

Example 3: Efficient micro-payments

lock_time


```
{
    "hash":"5a42590...b8b6b",
     "ver":1,
     "vin_sz":2,
     "vout_sz":1,
     "lock_time":315415,
     "size":404,
...
}
```


Block index or real-world timestamp before which this transaction can't be published

Bitcoin blocks

Bitcoin blocks

Why bundle transactions together?

- Single unit of work for miners
- Limit length of hash-chain of blocks
  - Faster to verify history

Bitcoin block structure

Hash tree (Merkle tree) of transactions in each block

The real deal: a Bitcoin block

block header

transaction data


```
{
 "hash":"00000000000000001aad2...",
 "ver":2,
 "prev_block":"00000000000000003043...",
 "time":1391279636,
 "bits":419558700,
 "nonce":459459841,
 "mrkl_root":"89776...",
 "n_tx":354,
 "size":181520,
 "tx":[
  ...
 ],
 "mrkl_tree":[
  "6bd5eb25...",
  ...
  "89776cdb..."
 ]
}
```

The real deal: a Bitcoin block header

mining puzzle information


```
{
 "hash":"0000000000000001aad2...",
 "ver":2,
 "prev_block":"00000000000000003043...",
 "time":1391279636,
 "bits":419558700,
 "nonce":459459841,
 "mrkl_root":"89776...",
 ..
}
```


hashed during mining

not hashed

The real deal: coinbase transaction

```
"in":[
    {
      "prev_out":{
        "hash":"000000.....0000000",
        "n":4294967295
      },
    "coinbase":"..."
    },
   "out":[
 {
   "value":"25.03371419",
   "scriptPubKey":"OPDUP OPHASH160 ... "
    }
```

redeeming nothing

arbitrary

Null hash pointer

First ever coinbase parameter:
"The Times 03/Jan/2009 Chancellor on brink of second bailout for banks"

block reward

transaction fees

See for yourself!

Transaction View information about a bitcoin transaction

151b750d1f13e76d84e82b34b12688811b23a8e3119a1cba4b4810f9b0ef408d

1KryFUt9tXHvaoCYTNPbqpWPJKQ717YmL5 → 1KvrdrQ3oGqMAiDTMEYCcdDSnVaGNW2YZh 1.0194 BTC
1KryFUt9tXHvaoCYTNPbqpWPJKQ717YmL5 3.458 BTC

9 Confirmations | 4.4774 BTC

Summary	
Size	257 (bytes)
Received Time	2014-08-05 01:55:25
Included In Blocks	314018 (2014-08-05 02:00:40 +5 minutes)
Confirmations	9 Confirmations
Relayed by IP	Blockchain.info
Visualize	View Tree Chart

Inputs and Outputs	
Total Input	4.4775 BTC
Total Output	4.4774 BTC
Fees	0.0001 BTC
Estimated BTC Transacted	1.0194 BTC
Scripts	Show scripts & coinbase

blockchain.info (and many other sites)

The Bitcoin network

Bitcoin P2P network

- Ad-hoc protocol (runs on TCP port 8333)
- Ad-hoc network with random topology
- All nodes are equal
- New nodes can join at any time
- Forget non-responding nodes after 3 hr

Joining the Bitcoin P2P network

Transaction propagation (flooding)

Should I relay a proposed transaction?

- Transaction valid with current block chain
- (default) script matches a whitelist
  - Avoid unusual scripts
- Haven't seen before
  - Avoid infinite loops
- Doesn't conflict with others I've relayed
  - Avoid double-spends

Sanity checks only...
Some nodes may ignore them!

Nodes may differ on transaction pool

Race conditions

Transactions or blocks may *conflict*

- Default behavior: accept what you hear first
- Network position matters
- Miners may implement other logic!

Block propagation nearly identical

Relay a new block when you hear it if:

- Block meets the hash target
- Block has all valid transactions
  - Run *all* scripts, even if you wouldn't relay
- Block builds on current longest chain
  - Avoid forks

Sanity check
Also may be ignored...

Block propagation Time

Source: Yonatan Sompolinsky and Aviv Zohar: “Accelerating Bitcoin’s Transaction Processing” 2014

How big is the network?

- Impossible to measure exactly
- Estimates-up to 1M IP addresses/month
- Only about 5-10k “full nodes”
    - Permanently connected
    - Fully-validate
- This number may be dropping!

Bitcoin full node distribution

RANK	COUNTRY	NODES
1	United States	2466 (24.32%)
2	Germany	1936 (19.09%)
3	France	674 (6.65%)
4	Netherlands	484 (4.77%)
5	China	402 (3.96%)
6	Canada	402 (3.96%)
7	United Kingdom	351 (3.46%)
8	Singapore	312 (3.08%)
9	Russian Federation	270 (2.66%)
10	Japan	248 (2.45%)
	More (102)	

TOTAL 10140 NODES AT 5 PM ON JAN 23, 2019

Fully-validating nodes

- Permanently connected
- Store entire block chain
- Hear and forward every node/transaction

Storage costs

Tracking the UTXO set

- **U**nspent **T**ransaction **O**utput
  - Everything else can be stored on disk
- Currently ~61.7 M UTXOs
  - Out of 375 M transactions (as of Jan 2019)

Thin/SPV clients (not fully-validating)

Idea: don’t store everything

- Store block headers only
- Request transactions as needed
  - To verify incoming payment
- Trust fully-validating nodes

1000x cost savings! (200 GB->200MB)

Software diversity

- About 90% of nodes run “Core Bitcoin” (C++)
  - Some are out of date versions
- Other implementations running successfully
  - BitcoinJ (Java)
  - Libbitcoin (C++)
  - btcd (Go)
- “Original Satoshi client”

Limitations & improvements

Hard-coded limits in Bitcoin

- 10 min. average creation time per block
- 1 M bytes in a block (pre SegWit)
- 20,000 signature operations per block
- 100 M *satoshis* per bitcoin
- 21M total bitcoins maximum
- 50,25,12.5... bitcoin mining reward

These affect economic balance of power too much to change now

Throughput limits in Bitcoin

- 1 M bytes/block (10 min) - post SegWit is slightly different
- >250 bytes/transaction
- 7 transactions/sec ☹

Compare to:

- VISA: 2,000-10,000 transactions/sec
- PayPal: 50-100 transaction/sec

Cryptographic limits in Bitcoin

- Only 1 signature algorithm (ECDSA/P256)
- Hard-coded hash functions

Crypto primitives might break by 2040...

“Hard-forking” changes to Bitcoin

Soft forks

Observation: we can add new features which only *limit* the set of valid transactions

Need majority of nodes to enforce new rules

Old nodes will approve

RISK: Old nodes might mine now-invalid blocks

Soft fork example: pay to script hash

<signature>
<<pubkey> OP_CHECKSIG>

OP_HASH160
<hash of redemption script>
OP_EQUAL

Old nodes will just approve the hash, not run the embedded script

Soft fork possibilities

- New signature schemes
- Extra per-block metadata
  - Shove in the coinbase parameter
  - Commit to UTXO tree in each block

Hard forks

- New op codes
- Changes to size limits
- Changes to mining rate
- Many small bug fixes

Blockchain Technology And Applications

Sandeep K. Shukla

IIT Kanpur

C3I Center

Acknowledgement

- The material of this lecture material is from various websites in particular related to Truffle and Ganache-Cli (including:
- https://medium.com/haloblock/deploy-your-own-smart-contract-with-truffle-and-ganache-cli-beginner-tutorial-c46bce0bd01e
- https://github.com/ethereum/wiki/wiki/Design-Rationale
- https://blockgeeks.com/guides/solidity/
- https://truffleframework.com/tutorials/ethereum-overview

Introduction to Ethereum

Revisit Blockchain

- So far, with bitcoin, we saw use of blockchain as a
  - Creation engine for digital currency
  - A distributed, tamper-resistant log of transactions in crypto-currency
  - A distributed ledger that solves heuristically Byzantine-safe distributed consensus
- Now, we look into more generic blockchain
  - That can be thought of as a distributed execution engine for consistent program execution
  - That can also support crypto-currency and in fact uses crypto-currency to solve consensus problem
  - That can be used to enforce contracts between parties through smart contracts

Agenda of this lecture

- Relook at the Blockchain technology
  - Why use a blockchain?
  - What is a blockchain?
  - How does a blockchain work?
- The Ethereum Blockchain
  - What is Ethereum?
  - What is a smart contract?
  - Ethereum Networks
  - Distributed Applications (Dapps)
- Truffle and Ganache-Cli for your first smart contract

Why use Blockchain?

- Blockchains are used when multiple parties, perhaps located across the world, need to share data and transfer value without trusting each other.
- The financial world describes this trust as the **counterparty risk**:
  - ***the risk that the other party won't hold up their end of the bargain.***
- Blockchains attempt remove the counterparty risk through a clever usage of mathematics, cryptography, and peer-to-peer networking.

Classical Database Applications

centralized approach

- Can be manipulated from inside and outside
- We have to trust the owners of the database and servers to keep data secure and with integrity
- Hackers can also infiltrate the server and change data
- Centralized backup and restore can not be necessarily trusted

Mitigating security/integrity issues in Centralized Data stores

- Every time data changes, make a backup copy and retain all historical backups
  - Hash the backup and keep it safe to prove integrity violation
- To share data, all stake holders must agree that the data is not tampered with (some proof mechanism might be required)
- Only way to ensure all that is through trusted 3rd party audit

What is a block chain in this context?

- Shared database consisting of ledger of transactions
- Every stake holder keeps a copy of the ledger and can verify all transactions that are put in the ledger
- Reading/writing on the ledger is completely decentralized and secure
- Fault tolerance
- Independent verification by anyone interested : disintermediation (anyone can audit)

How does Blockchain work?

- Nodes, Transactions, Blocks
- Mining through solving hard problems (solving Byzantine fault-tolerant consensus)
- Hashing for integrity
- Digital Signature for Authenticity and/or authorization
- Permanence (Tamper resistance)

Blockchain in Pictures

Bob attempts to send Alice 1 ETH

Bob and Alice's transaction is combined with other transactions that have occurred since the last block

Blockchain in Pictures (2)

Miners compete to validate the block with the new set of transactions

The victorious miner creates a new block and receives a reward

With the transaction validated, Alice receives 1 ETH

What is Ethereum?

- **Ethereum is a blockchain that allows you to run programs in its trusted environment.**
  - contrasts with the Bitcoin blockchain, which only allows you to manage cryptocurrency.
- Ethereum has a virtual machine -- Ethereum Virtual Machine (EVM).
- The EVM allows code to be verified and executed on the blockchain,
  - providing guarantees it will be run the same way on everyone's machine.
- This code is contained in "smart contracts"
- Ethereum maintains the state of the EVM on the blockchain.
  - All nodes process smart contracts to verify the integrity of the contracts and their outputs.

What is a smart contract?

- **A smart contract is code that runs on the EVM.**
- Smart contracts can accept and store ether, data, or a combination of both.
- Using the logic programmed into the contract,
  - it can distribute that ether to other accounts or even other smart contracts.
- Example:
  - Alice wants to hire Bob to build her a patio
  - they are using an escrow contract (a place to store money until a condition is fulfilled) to store their ether before the final transaction.

Smart Contract in Pictures

Language of Smart Contracts in Ethereum

- Smart Contracts for Ethereum are written in Solidity
  - Solidity is statically typed
  - supports inheritance, libraries, and complex user-defined types
  - Similar to Javascript syntactically

  - To learn solidity go to https://remix.ethereum.org and you can start programming smart contracts without having to create your own Ethereum network

Some basic Ideas in Solidity

- Meant to execute as bytecode after compilation on Ethereum EVM
- EVM has a stack and Memory Model
  - 32 byte instruction word size
  - Access to program “stack” – like a register space where memory addresses may be stored to make program counter loop/jump
  - An expandable temporary “memory”
  - More permanent “storage” which is actually written into permanent block chain as program states
  - NO non-determinism allowed (e.g., no random() like function calls)

Program Execution

- When an Ethereum block is “mined”,
  - the smart-contract deployments and function calls within that block get executed on the node that mines the block
    - the new state changes to any storage spaces or transactions within that smart-contract actually occur on that miner node.
  - As the new block gets propagated to all the other nodes
    - Each node tries to independently verify the block,
    - Verifying includes doing those same state changes to their local copy of the blockchain
  - it will fail if the smart-contract acts non-deterministically.
    - If the other nodes cannot come to a consensus about the state of blockchain after the new block and its contracts get executed, the network could literally halt.

Program Execution (2)

- EVM smart-contracts cannot access data outside the “memory”, and “storage”
  - (we don’t want the smart-contract to be able to read or delete the hard-drives of the nodes it runs on)
- Cannot query outside resources like with a JQuery.
- Do not have access to many library functions like for parsing JSON structures or doing floating-point arithmetic,
  - it’s actually cost-prohibitive to do those sub-routines or store much data in the Ethereum blockchain itself.

Program Execution (3)

- When you call a smart-contract that does some state-changing work or computation, you will incur a **gas "cost"** for the work done by the smart contract
  - this gas cost is related to the amount of computational work required to execute your function.
  - sort of a "micropayment for microcomputing" system, where you can expect to pay a set amount of gas for a set amount of computation, forever.
- The price of gas itself is meant to stay generally constant, meaning that when Ether goes up on the global markets, the **price** of gas against Ether should go down.
- When you execute a function call to a smart-contract, you can get an estimation of the amount of gas you must pay beforehand, but you must also specify the **price** (in ether per gas) that you are willing to pay
  - the mining nodes can decide if that's a good enough rate for them to pick up your smart-contract function call in their next block.

Addresses of Solidity Smart Contracts

- Smart-contracts have their own address, from which they can receive and send Ether.
- Smart contracts can track the “caller” of the function in a verifiable way,
  - it can determine if one of its functions is being called by a privileged “owner” or “admin” account, and act accordingly for administrative functions.
- They have the ability to read data from the Ethereum blockchain, and access info on transactions in older blocks.

Getting data outside of Blockchain

- But are smart-contracts “locked in” into their own little deterministic world, only able to be aware of data stored in the Ethereum blockchain itself?
  - We can make a call to an oracle that will tell us something about the outside world in a trustable way, and *act on that data* within the smart contract.
  - even though real-world events themselves are not deterministic, the Oracle can be trusted to always answer every node’s request about what happened in a deterministic way
    - so that all nodes can still come to a consensus.
- An “oracle” will take some data, say a ticker price feed about a real-world stock price, and record that data into “storage” in a simple Oracle smart-contract,

Ethereum networks

- On the **MainNet**, data on the chain—including account balances and transactions—are public, and anyone can create a node and begin verifying transactions.
- Ether on this network has a market value and can be exchanged for other cryptocurrency or fiat currencies like US Dollars.
- But there are other Ethereum networks as well.

Other Ethereum Networks

- The Ethereum blockchain can be simulated locally for development.
- Local test networks process transactions instantly and Ether can be distributed as desired.
- An array of Ethereum simulators exist;
  - Ganache
- Developers use public test networks (or testnets) to test Ethereum applications before final deployment to the main network.
  - Ether on these networks is used for testing purposes only and has no value.

Public TestNets

- **Ropsten**: The official test network, created by The Ethereum Foundation. Its functionality is similar to the MainNet.
- **Kovan**: A network that uses a consensus method called "proof-of-authority".
  - This means its transactions are validated by select members, leading to a consistent four second block time.
  - The supply of ether on this testnet is also controlled to mitigate spam attacks.
- **Rinkeby**: A testnet also using proof-of-authority, created by The Ethereum Foundation.
- **Gorli:** Another proof-of-authority testnet.

Private/Enterprise Networks

- Private Ethereum networks allow parties to share data without making it publicly accessible.
- A private blockchain is a good choice for:
  - Sharing of sensitive data, such as health care records
  - Scaling to handle higher read/write throughput, due to the smaller network size
- An example of a private enterprise blockchain is Quorum, originally written by J.P. Morgan.

Distributed Applications (Dapps)

- **Applications using smart contracts for their processing are called "distributed applications", or "dapps".**
- The user interfaces for these dapps consist of familiar languages such as HTML, CSS, and JavaScript.
- The application itself can be hosted on a traditional web server or on a decentralized file service such as Swarm or IPFS.
- Dapps based solution available for:
  - Record keeping
  - Finance
  - Supply chains
  - Real estate
  - Marketplaces

Ethereum Design Principles

- **Sandwich Complexity Model**
  - Complexity is handled by high-level-language compilers, argument serialization and deserialization scripts, storage data structure models, the leveldb storage interface and the wire protocol,
- **Freedom (inspired by net neutrality)**
  - users should not be restricted in what they use the Ethereum protocol for, and the designers should not attempt to preferentially favor or disfavor certain kinds of Ethereum contracts or transactions based on the nature of their purpose.
- **Generalization**
  - protocol features and opcodes in Ethereum should embody maximally low-level concepts, so that they can be combined in arbitrary ways including ways that may not seem useful today but which may become useful later,
- **Low in high level features:**
  - a corollary to generalization, the Ethereum foundation often refuses to build in even very common high-level use cases as intrinsic parts of the protocol, with the understanding that if people really want to do it they can always create a sub-protocol (eg. ether-backed subcurrency, bitcoin/litecoin/dogecoin sidechain, etc) inside of a contract.

Some Differences with Bitcoin blockchain

- Accounts and not UTXOs
- Merkle Patricia Trees
- RLP – Recursive Length Prefix – main serialization format
- Compression Algorithm
- Trie Usage
- Uncle Incentivization
- Difficulty Update Algorithm
- Gas and Fees
- EVM

Accounts and not UTXOs

- Bitcoin stores data about users' balances in a structure based on *unspent transaction outputs* (UTXOs):
  - the entire state of the system consists of a set of "unspent outputs”
    - such that each coin has an owner and a value, and a transaction spends one or more coins and creates one or more new coins
- Transaction validity
  - Every referenced input must be valid and not yet spent
  - The transaction must have a signature matching the owner of the input for every input
  - The total value of the inputs must equal or exceed the total value of the outputs
- A user's "balance" in the system is thus the total value of the set of coins for which the user has a private key capable of producing a valid signature.

Accounts vs UTXOs

- In Ethereum, the state stores a list of accounts
  - each account has a balance, as well as Ethereum-specific data (code and internal storage)
- a transaction is valid if the sending account has enough balance to pay for it,
  - the sending account is debited and the receiving account is credited with the value.
- If the receiving account has code
  - the code runs, and internal storage may also be changed,
  - or the code may even create additional messages to other accounts which lead to further debits and credits

Accounts vs UTXOs

- Benefits of UTXOs
  - **Higher degree of privacy**: if a user uses a new address for each transaction that they receive then it will often be difficult to link accounts to each other
  - **Potential scalability paradigms**:
    - UTXOs are more compatible with certain kinds of scalability paradigms,
    - we can rely on only the owner of some coins maintaining a Merkle proof of ownership,
    - if everyone including the owner decides to forget that data then only the owner is harmed.

Accounts vs. UTXOs (Benefits of Accounts)

- **Large space savings**:
  - if an account has 5 UTXO, then switching from a UTXO model to an account model would reduce the space requirements
  - from (20 + 32 + 8) * 5 = 300 bytes (20 for the address, 32 for the txid and 8 for the value) to 20 + 8 + 2 = 30 bytes (20 for the address, 8 for the value, 2 for a nonce)
  - Transactions can be smaller (e.g., 100 bytes in Ethereum vs. 200-250 bytes in Bitcoin) because
    - every transaction need only make one reference and one signature and produces one output.

Benefits of Accounts (2)

- **Greater fungibility**:
  - there is no blockchain-level concept of the source of a specific set of coins, it becomes less practical to institute a redlist/blacklisting scheme
  - Difficult to draw a distinction between coins depending on where they come from.
- **Simplicity**:
  - easier to code and understand, especially once more complex scripts become involved.
- **Constant light client reference**:
  - light clients can at any point access all data related to an account by scanning down the state tree in a specific direction.
- **One problem is that in order to prevent replay attacks,**
  - every transaction must have a "nonce", such that the account keeps track of the nonces used and only accepts a transaction if its nonce is 1 after the last nonce used.

Patricia Tree

Prefix tree/Radix tree/Trie

- Trie uses a key as a path so the nodes that share the same prefix
- This structure is fastest at finding common prefixes, and requires small memory.
- commonly used for implementing routing tables, systems that are used in low specification machines like the router.

Merkle Tree

Top Hash
hash(Hash 0 + Hash 1)
Hash 0
hash(Hash 0-0 + Hash 0-1)
Hash 1
hash(Hash 1-0 + Hash 1-1)
Hash 0-0
hash(L1)
Hash 0-1
hash(L2)
Hash 1-0
hash(L3)
Hash 1-1
hash(L4)
L1
L2
L3
L4
Data Blocks

Merkle Patricia Tree (MPT)

Now some actual work (assuming ubuntu)

- > sudo apt-get update
- > sudo apt-get install nodejs
- > sudo apt-get install npm
- > npm –v (make sure 5.0+)
- > sudo npm install –g truffle
- > sudo npm install –g ganache-cli
- > sudo ganache-cli

```
william@william-VirtualBox:~$ ganache-cli
Ganache CLI v6.1.3 (ganache-core: 2.1.2)

Available Accounts
==================
(0) 0xa6df0f09b87787cb8532ac90d603600708ddca23
(1) 0x0f3980a037b5bc2c55b4bf20ef519f3e51847a4e
(2) 0xf2c2c2df9acf60b20b9bb4baa4108821a4a7f6ba
(3) 0x2be6125a439342d7a5f767957fa7ca9c9be357de
(4) 0x9e3e2240dca5e02284759438b05dcaa535e0f166
(5) 0x03c2b9f7275c2fba0c90aeb24cf0660d06908b30
(6) 0xb75c95dbbb359526cbdac8bfcc3d33d3fc4006c0
(7) 0x51509f625bc11638826d1349aa142e95c4f6c003
(8) 0x37c5d4abfdd92d56ce21f5cfcacb907f49f4a08d
(9) 0x24d2b74e63b83382750a53a0870977c14ccbdf07

Private Keys
==================
(0) c014d77f61a413d909d5d33142013b3c924085364de618aa5b94c604bf092770
(1) ebbff0e4394a42632683d2fd5ed33d31dfbc51f83f5c09be4f0d3eecbf91eb71
(2) 5ed898e70aaeaf1d1a148432d67c89b9707e94ac830958c14961ed4034eb1e4e
(3) d73735195314d1885a173f92772240780b93b8769f57dd218ae8170b2283272b
(4) 80165a637888ad8e95e14b78f625b61087ad5a80cf7ab20c6fa7ff765acef176
(5) 80c42d081bb6b1a0d25a91c23010142b67d210820d8804adbd0cd0fc17806163
(6) e9d661c3b05d05588f78ef9f2e35faacc874f2ca14dd30d887dd90e1635c9018
(7) f90ebc1c0ff8fa08305dee613aaf65e51a2d09d7f9d19d1cf1a0371a5ff1f72a
(8) 2b9af99fb51f4ced631091f6392e80fb24185abedeabb1e85af1a95b5ee52f8e
(9) af1defd9a5cb596b2982481e804699633b99a84057100be9dbe0a94ce635bbb1

HD Wallet
==================
Mnemonic:      rookie mistake degree announce episode bright result nation tomorrow pond outer force
Base HD Path:  m/44'/60'/0'/0/{account_index}

Gas Price
==================
20000000000

Gas Limit
==================
6721975

Listening on localhost:8545
```

HelloWorld smart contract

- > mkdir SmartContractDemo
- > cd SmartContractDemo
- > mkdir HelloWorld
- > cd HelloWorld
- > truffle init

Truffle init

```
william@william-VirtualBox:~/ethereum$ mkdir SmartContractDemo
william@william-VirtualBox:~/ethereum$ cd SmartContractDemo/
william@william-VirtualBox:~/ethereum/SmartContractDemo$ mkdir HelloWorld
william@william-VirtualBox:~/ethereum/SmartContractDemo$ cd HelloWorld/
william@william-VirtualBox:~/ethereum/SmartContractDemo/HelloWorld$ truffle init
Downloading...
Unpacking...
Setting up...
Unbox successful. Sweet!

Commands:

  Compile:        truffle compile
  Migrate:        truffle migrate
  Test contracts: truffle test
william@william-VirtualBox:~/ethereum/SmartContractDemo/HelloWorld$ ls
contracts  migrations  test  truffle-config.js  truffle.js
william@william-VirtualBox:~/ethereum/SmartContractDemo/HelloWorld$
```

Effect of unbox

Directory structure created by unbox

- ***/contracts***: store original codes of the smart contract. We will place our HelloWorld.sol file here.
- ***/migrations***: deploy the smart contract in the "*contracts*" folder.
- ***/test***: test codes for your smart contract, support both JavaScript and Solidity.
- ***truffle.js***: configuration document.
- ***truffle-config.js***: configuration document for windows user. We can just leave it alone.

  if your Ubuntu 16.04 is running on a virtual machine under Windows, then your configuration should also go here

Creating contract

- > truffle create contract HelloWorld

```
william@william-VirtualBox:~/ethereum/SmartContractDemo/HelloWorld$ truffle create contract HelloWorld
william@william-VirtualBox:~/ethereum/SmartContractDemo/HelloWorld$ cd contracts/
william@william-VirtualBox:~/ethereum/SmartContractDemo/HelloWorld/contracts$ ls
HelloWorld.sol  Migrations.sol
william@william-VirtualBox:~/ethereum/SmartContractDemo/HelloWorld/contracts$
```

HelloWorld.sol

```
pragma solidity ^0.5.0;

Contract HelloWorld {

    function hi() public pure returns (string memory) {

        return (“Hello World”);

    }

}
```

> truffle compile

Compiling and deploying contract

- Compilation creates bytecode for EVM
- .json file in build/contracts directory as HelloWorld.json
- Now time to deploy the contracts
- Create a new file under migrations directory
  - 2_deploy_contracts.js

```
var HelloWorld=artifacts.require (“./HelloWorld.sol”);

module.exports = function(deployer) {

    deployer.deploy(HelloWorld);

}
```

Configuring to deploy on local Ethereum network

```
15    module.exports = {
16      networks: {
17        development: {
18          host: "localhost",
19          port: 8545
20        }
21      }
22    };
23
```

> truffle migrate

Migration in work

```
william@william-VirtualBox:~/ethereum/SmartContractDemo/HelloWorld$ truffle migrate
Using network 'development'.

Running migration: 1_initial_migration.js
  Deploying Migrations...
  ... 0x942a4817ab47ebdbae62d86552656f427f5efb92b60d67ce2afb86963d5d4fa0
  Migrations: 0x84992eff29fbac77a60f5470e48ba767718b8f6c
Saving successful migration to network...
  ... 0x4dcd3528ba8ee21ec2e81cac9e2aabe209b53fe03000160c80e83e853c4df6c4
Saving artifacts...
Running migration: 2_deploy_contracts.js
  Deploying HelloWorld...
  ... 0x9b1066d86effeba988c111542f2d1ae55864fa6d9bc017708ba7850bd43f0e40
  HelloWorld: 0xd679d052b3eeefb041d9393fb9ceeb7145b0a173
Saving successful migration to network...
  ... 0x1c6a33d2cb74baf0b0e51ecfffe5b79a78de636d949715fc7e85b33bd591fc62
Saving artifacts...
```

Effect of the deployment on the network

```
net_version
eth_accounts
eth_accounts
net_version
net_version
eth_sendTransaction

  Transaction: 0x942a4817ab47ebdbae62d86552656f427f5efb92b60d67ce2afb86963d5d4fa0
  Contract created: 0x84992eff29fbac77a60f5470e48ba767718b8f6c
  Gas usage: 277462
  Block Number: 1
  Block Time: Fri Jun 22 2018 16:39:21 GMT-0700 (PDT)

eth_newBlockFilter
eth_getFilterChanges
eth_getTransactionReceipt
eth_getCode
eth_uninstallFilter
eth_sendTransaction

  Transaction: 0x4dcd3528ba8ee21ec2e81cac9e2aabe209b53fe03000160c80e83e853c4df6c4
  Gas usage: 42008
  Block Number: 2
  Block Time: Fri Jun 22 2018 16:39:22 GMT-0700 (PDT)

eth_getTransactionReceipt
eth_accounts
net_version
net_version
eth_sendTransaction

  Transaction: 0x9b1066d86effeba988c111542f2d1ae55864fa6d9bc017708ba7850bd43f0e40
  Contract created: 0xd679d052b3eeefb041d9393fb9ceeb7145b0a173
  Gas usage: 136299
  Block Number: 3
  Block Time: Fri Jun 22 2018 16:39:22 GMT-0700 (PDT)
```

Summary

- In this lecture, you learnt the design philosophy of Ethereum Blockchain
- Some Basic differences between Bitcoin blockchain and Ethereum blockchain
- The concept of smart contracts
- The concept of Ethereum network and simulation of such networks for testing
- Finally, clue about how to simulate a local private network, and try out a simple smart contract on your own

A Study of Inequality in the Ethereum Smart Contract Ecosystem

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY KANPUR

An Analysis of Ethereum and cryptocurrency Blockchain

- Motivation
- Collection of on-chain smart contracts
- Analysis of on-chain smart contracts - identification of the '*Contracts of Importance*'
- Results of the tools based analysis on these contracts

Recall

- Ethereum is the **second most valuable** cryptocurrency
- It is the brainchild of **Vitalik Buterin** who wrote the white paper
- Ethereum also uses PoW (migrating to Casper soon)
- Ethereum can also store and run small computer programs called smart contracts
- The Ethereum Protocol is maintained by the Ethereum Foundation
- Unlike Bitcoin, Ethereum has two kinds of addresses -
  - Externally Owned Accounts (EOA)
  - Contract Accounts

Image Source - https://commons.wikimedia.org/

Recall (2)

- The term smart contracts was coined by **Nick Szabo** in 1997
- They are small programs that exist on the blockchain and are executed by the Ethereum Virtual Machine (EVM).
- All the functionality required is implemented in the smart contract logic, and since the code itself resides on the blockchain it becomes highly tamper resistant.
- This property is crucial in applications like escrow payments, voting systems, etc.
- Finding new application areas for smart contracts is an active research area.

Recall: Ethereum Virtual Machine (EVM)

- Provides a layer of **abstraction** between the code and the machine
- Also makes the code **portable** across different machines
- The EVM has **140 opcodes** which allow it to be **Turing complete**
- Each opcode takes 1 byte of storage space.
- The EVM also uses a 256 bit register stack which holds 1024 items.
- It also has a contract memory (non-persistent) for complicated operations
- For storing data indefinitely, storage is used.
- Reading from storage is free, but writing to storage is extremely expensive.

```
pragma solidity 0.4.25;

contract SimpleStorage {
uint storedData;

    function set(uint x) public {
        storedData = x;
    }
    function get() public view returns (uint retVal) {
        return storedData;
    }
}
```

solc


```
6080604052348015610010576000080f
d5b5060df8061001f6000396000f300
6080604052600436106049576000035
7c0100000000000000000000000000
0000000000000000000000000000009
00463ffffffff16806360fe47b114604e5
780636d4ce63c146078575b600080fd
5b348015605957600080fd5b5060766
0048036038101908080359060200 19
09291905050506 0a0565b005b34801
5608357600080fd5b50608a60aa565b
6040518082815260200191505060 40
5180910390f35b80600081905550505
65b600080549050905600a165627a7
a723058201e222f69bce1d87a752237
78a5ba301b0a4490d9a589c0fee6df4
8a067d3e8ac0029
```


EVM


```
6080604052600436106049576000035
7c0100000000000000000000000000
0000000000000000000000000000009
00463ffffffff16806360fe47b114604e5
780636d4ce63c146078575b600080fd
5b348015605957600080fd5b5060766
0048036038101908080359060200 19
09291905050506 0a0565b005b34801
5608357600080fd5b50608a60aa565b
6040518082815260200191505060 40
5180910390f35b80600081905550505
65b600080549050905600a165627a7
a723058201e222f69bce1d87a752237
78a5ba301b0a4490d9a589c0fee6df4
8a067d3e8ac0029
```


- solidity code
- contract creation code
- runtime byte-code (resides on the blockchain)

Solidity

- Solidity is the most popular **programming language** for Ethereum Smart Contracts
- It is similar to Javascript and C++
- Other experimental languages like Vyper and Bamboo are not much in use
- Solidity Compiler (**`solc`**) converts the source code to EVM bytecode

Image Source - https://en.wikipedia.org/wiki/Solidity

Ether and Gas

- **Ether** is the native cryptocurrency of the Ethereum Network
- **Gas** is a unit to measure the computational work done.
  - Introduced as it would have been unfair to base the transaction fees on just the transaction length or keep it constant
  - Each operation has a **gas cost**
  - Every transaction mentions the **gas price**
  - The two together give the transaction fees in Ether
  - Two new scenarios -
    - Transaction running out of gas
    - Gas price too low/high

Collection of On-Chain Smart Contracts

Collection of Smart Contracts - The Challenges

- **How to get the smart contract data?**
  - The most direct way is to connect to the Ethereum Network using a client like `geth` and run a **full node** that is usually run by miners.
  - However, the Ethereum Blockchain data had crossed **1TB** in May, 2018.
  - Syncing a full Ethereum node takes a lot of **time**
  - Therefore, running a full node was infeasible on the IITK network

- **Which addresses contain smart contracts?**
  - Ethereum does not distinguish between addresses in use (by smart contracts or users) or orphaned addresses which are yet to be claimed.
  - There are **16^{40} addresses** on Ethereum network - brute-forcing them was not feasible

Collection of Smart Contracts - Our Approach

- We leveraged the **INFURA** API (by Consensys) and leveraged its *'geth-like'* methods (**`getCode()`**) and **Web3** to get the smart contract data.

```
for blockNo 1 to 7.1M do:
        for all txn in blockNo do:
                store from_addr and to_addr in addr_file.txt

sort -o addr_file_u.txt -u addr_file.txt

for all addr in addr_file_u.txt do:
        code = geth.getCode(addr)
        if(code != 0x0):
                store code in addr.bin.hex
```

Tradeoff: Only contracts that are live & have been interacted with

- However, the **search space was huge** and the network became a bottleneck. Therefore, we distributed the workload accross **GCP Compute Engine** instances

Collection of Smart Contract Source Codes

- **Etherscan** gives an option to smart contract developers to publish their source codes of the on-chain contracts.
- This allows for better transparency in the system as the **readability** and **auditability** of the source codes is much easier.
- For all the collected smart contract addresses, we call the etherscan APIs `getsourcecode` endpoint, and store the solidity code if available.

Collection of Smart Contracts - Summary

Final collected dataset -

- **1.9M** smart contract bytecodes
- **887K** smart contract source codes

Analysis of On-Chain Smart Contracts

Analysis on the smart contracts - Parameters

DUPLICITY

ETHER BALANCE

NUMBER OF TRANSACTIONS

VALUE OF TRANSACTIONS

CONTRACT CREATION

* For all categories the results were aggregated for duplicate smart contracts

Duplicity

- It is observed that out of 1.9M smart contracts, **1.8M (94.6%)** are duplicates
- The most duplicated contract is User Wallet contract with over **650K** instances
- However, only 2 out of the top 10 have verified source codes available
- It is observed that the **top hundred** contracts (0.1%) amount to a total of **1.72M** (90.28%) occurrences in the dataset
- We call these **High Occurrence Targets**

Contract Bytecode Hash	src code	Contract Name/ Owner	Freq
2bf69ddcf80f6b24f2e6a8bf1454f662	Y	User Wallet	651930
fa00c5b8d83dbf920aec56d52c1df224	N	?	158186
55f0329f9e5dbac461e933c66e0e29b5	N	?	115132
dfcc91bcdc37abae7e8e9c82d57fbf6d	Y	Forwarder	99548
702edb219bba3238d55b2b38c759798b	N	?	90489
923d7eaf6e90eb272493d3ca5c5859d5	N	?	78018
7b63bae3ec81aa70d809a091240dccaa	N	?	42868
62dbffb5cce3d14500568320ab6dcd75	N	?	40456
1ae99eb3c89152c83cf788a5e7df4532	N	?	37534
125fb7c1ad488e0d0b9b034cfd12a977	N	?	28255

Ether Balance

- The collected contracts contain a total of 10.88M ETH (worth **US$1.66B**[1])
- The most valuable contract is Wrapped Ether (WETH9). It contains roughly 2.4M ether (22%).
- It is observed that the **top hundred** contracts (0.1%) contain **10.7M** (98.86%) ETH. This is worth **US$1.6B**[1]
- We call these **High Value Targets**

Contract Bytecode Hash	src code	Contract Name/ Owner	ETH Balance
0e10248e905a92ffd070393ea0a2890c	Y	WETH9	2383501
e6bf83c2201a1b2070a529a4f4814488	Y	Wallet	1430029
d62d28a4c3fad57a5158297ce6efed9e	N	Gemini's Cold Wallet	895999
108810ec1a9185e325c05b24f1a1c21e	N	Ethereum Foundation	645173
f5c40e048aca031a2ea4f32ba04646e1	Y	MultiSigWalletWithDailyLimit	600341
c7010bf53217a9fe2fc6ae82cf19907d	Y	MultiSigWalletWithDailyLimit	568083
4143e0500473d8164fae03423612e9e4	Y	Wallet	515035
c5fa4304659be1268f19e04c1a4add89	Y	MultiSigWallet	395432
cdb19b39b2dc549cd112a1a9ca3197ac	Y	MultiSigWallet	368023
1f086f1ded7ac3799011e0d6526bc436	N	?	292524

[1] as per exchange rate on 26th April, 2019

Number of Transactions

- Smart contracts in our data-set were involved in **175M** transactions (46% of all the transactions)
- **434K** contracts (22.7%) had only one recorded transaction on the blockchain
- Most interacted-with contract is the **User Wallet**
- We observe that **2500 contracts** (2.5%) amount to a total of **90.37%** (158M txns) of the total transactions done with smart contracts.
- We call these **High Interaction Targets**

Contract Bytecode Hash	src code	Contract Name/ Owner	Number of Txns
2bf69ddcf80f6b24f2e6a8bf1454f662	Y	UserWallet	5833642
91f778605c0976e25dc93fc4a591bb96	Y	EtherDelta	5272000
de379d9a60e5f52e45c99874eb61bc70	Y	IDEX Exchange	3966000
f779bf5757253c974724c46b1e9f441e	Y	KittyCore (CryptoKitties)	3141000
f7e3b0272ca30480eb26b89d198f4c84	Y	DSToken (EOS)	2955507
ee302cfe0db1c206484facb2c9543751	Y	TronToken (Tronix)	1990664
748f8df24378a8d1dd82d44e598c46f4	Y	HumanStandardToken	1904722
c24bf798c5a50007d907a5d397cc46b0	N	Poloneix Exchange	1766954
fa00c5b8d83dbf920aec56d52c1df224	N	?	1681369
ca200836bce3f6fcb8bc9bbd2b34b6c9	Y	Controller	1578593

Value of Transactions

- Smart Contracts in our dataset have been involved in transactions worth **484M ETH** (US$73B).
- **877K** contracts (45.9%) have not been involved in any transaction involving ETH.
- The most ether has been moved by **ReplaySafeSplit**
  - this was used heavily post the Ethereum hard fork in 2016 to avoid replay attacks.
- It is observed that the **top hundred** contracts (0.1%) have moved **459.4M ETH** (94.89%).
- We call these contracts **High Ether-Moving targets**

Contract Bytecode Hash	src code	Contract Name/ Owner	Value of Txns (in ETH)
9473b7b4e2820ec802fc3d3814f90916	Y	ReplaySafeSplit	85246694
ea63abcce55531452cf1f6fad33757cd	N	Bitfinex Exchange	50031528
c24bf798c5a50007d907a5d397cc46b0	N	Poloniex Exchange	42613873
b58e896a767147b204a8f9203c850c77	N	Kraken Exchange	41267671
70d285d1aa9cb3b94ce39ef82a59bf6d	N	Gemini Exchange	24808753
dc5b7eea9e5e2308d7efbfc6c33f4d96	N	?	21104195
e6bf83c2201a1b2070a529a4f4814488	Y	Wallet	18243092
aec1a8e9808dd257802994ae9bc737d4	Y	ReplaySafeSplit	15547424
709738b34faa8702cbd4568ff5c2e382	Y	DAO	11983614
108810ec1a9185e325c05b24f1a1c21e	N	Ethereum Foundation	11911040

Contract Creation Analysis

- Contracts can be created by
  - **Normal Transactions** - responsible for **39.4%** (753K) contract deployments.
  - **'Internal' Transactions -** not reflected on the blockchain. Amount to **60.6%** (1.16M) of all contract creations.
- The **first** contract was deployed on 7th August, 2015.
- Average gas used to deploy a contract is **318K** gas.
- The trend of contract deployments over time *closely* resembles the price graph of Bitcoin.

Contract Creation Analysis

- The **753K** contracts deployed by **normal** transactions, have been deployed by only **57,600** accounts.

Address	Total Contracts Deployed	Unique Contracts Deployed
0xb42b20ddbeabdc2a288be7ff847ff94fb48d2579	158189	4
0x9862d074e33003726fa05c74f0142995f33a3250	78018	1
0x42da8a05cb7ed9a43572b5ba1b8f82a0a6e263dc	57921	10
0x2e05a304d3040f1399c8c20d2a9f659ae7521058	40456	1
0x17bc58b788808dab201a9a90817ff3c168bf3d61	37534	1
0x866f649cd9280d3dfa282372a3f5828839944959	15272	1
0xe35f12181a2748285358b63cff25887410d0804b	14174	10
0xa2635b3d63b4e31976419865e1a81553bb347be3	13191	11
0x174443351e21d47ed9ab51517a301107d92ede64	13105	1
0x0536806df512d6cdde913cf95c9886f65b1d3462	12098	1

Contract Creation Analysis

- For the **1.16M** contracts deployed using **internal** transactions, it is observed that these contracts have been deployed by only **9228** unique contract addresses. This number reduces to **2420** contracts on considering duplicates.
- It is observed that the **top 100** (0.1%) contracts have deployed **1.14M** contracts **(98.93%)**
- We call these **High Origin Targets**

Contract Bytecode Hash	Total Contracts Deployed	Unique Contracts Deployed
b071cebdaac85e6cfa20b9de78314386	651930	1
23eed1c018699f2aa9217e487851262b	115132	1
07459966443977122e639cbf7804c446	99547	1
1409c3e3b055b70674d7446f3d30df36	90489	1
366dd689499a9ec1538b94d3c57bbcb0	62313	2
1093df22ad83b4efffcb608adcff6569	11369	1
11f7b022128ec8ceaa93375856a2613c	7904	1
05b915a788451c8a5f4d715914d8ca5c	7654	1
2d382304429d30b1706f4089d19dd265	6693	1
1a5383732e90b3fdbf70d13dd90b2684	6144	1

Analysis of on-chain Contracts - Summary

- We observe that the smart contracts on the blockchain are not as decentralised as expected.
- Across different categories, we observe that only a small fraction of smart contracts are 'important'.
- We call these **2900** (2715 unique) contracts - ***'Contracts of Importance'*** and use it for further studies.

Security of Ethereum

Security Issues

- DAO Attacks
- Ethereum is a public blockchain all the smart contracts are available
- To study the security trends and patterns in the real world smart contracts
- Analyse various smart contract verification tool on important contracts

Smart Contract Security - Why it is needed?

- Smart Contracts once deployed on the blockchain become **immutable** - as long as the blockchains integrity is not compromised.
- This attractive property becomes a unique challenge from a security and software engineering viewpoint -
  - new features and functionalities cannot be added
  - bugs and other vulnerabilities discovered after deployment cannot be patched
- It becomes imperative to thoroughly audit the smart contracts before deploying it and also develop new software development methodologies to cope up with these challenges

Impact of Smart Contract Insecurities

https://www.freecodecamp.org/news/a-hacker-stole-31m-of-ether-how-it-happened-and-what-it-means-for-ethereum-9e5dc29e33ce/

freeCodeCamp

20 JULY 2017

A hacker stole $31M of Ether — how it happened, and what it means for Ethereum

Gaming News

Ethereum Network Currently Marred by Ponzi Scheme-Like Games

Nick Marinoff on July 26, 2018 / 2 Comments

Post Views: 4,514

BUGS

Tron had a bug that could have made the entire network unusable

MAY 7, 2019, 11:46AM EDT

Unpatched Ethereum Clients Pose 51% Attack Risk, Says Report

Daniel Palmer

NEWS

https://www.theblockcrypto.com/linked/22366/tron-had-a-bug-that-could-have-made-the-entire-network-unusable

BY JOSIAH WILMOTH | OCTOBER 10, 2018 15:52 EST

Ethereum: We Haven't Seen the Last of the Bug That Killed the DAO

https://medium.com/@ogucluturk/the-dao-hack-explained-unfortunate-take-off-of-smart-contracts-2bd8c8db3562

The $280M Ethereum's Parity bug.

A critical security vulnerability in Parity multi-sig wallet got triggered on 6th November—paralyzing wallets created after the 20th July.

Matt Suiche Follow

Nov 7, 2017 · 3 min read

https://www.theregister.co.uk/2017/11/16/parity_flaw_not_fixed/

Problems faced by Developers and Users

- Since smart contracts is a new area of research, and industry is leading most of the efforts, not much information about smart contract vulnerabilities, tools and practices is available in the organized domain.
- The biggest problems are -
  - Studies on smart contract vulnerabilities are often **unorganized** and **incomplete**
  - Security tools do not have a corresponding **academic paper**
  - Most tools demonstrate their capabilities on **different** (sometimes **random**) **datasets**.
  - Also, they call similar vulnerabilities by **different names**
- This makes it impossible for a smart contract developer, or the end-user to make an informed decision about which tool to use and trust for their particular use-case.

Problems faced by Security Researchers

- Studies on smart contract vulnerabilities are often **outdated** and **incomplete**
- Lack of a **benchmark** - no way to compare the tools/research
- No information on the trends and patterns followed by the **real world smart contracts** - to guide security tool research and development

Insecurity Trends & Patterns

- According to the results of the tools on the **contracts of importance** the most common security vulnerabilities and issues are -
  - Re-entrancy https://medium.com/coinmonks/protect-your-solidity-smart-contracts-from-reentrancy-attacks-9972c3af7c21
  - Bad Randomness https://medium.com/dedaub/bad-randomness-is-even-dicier-than-you-think-7fa2c6e0c2cd
  - Denial of Service https://consensys.github.io/smart-contract-best-practices/known_attacks/
  - Visibility issues https://medium.com/loom-network/how-to-secure-your-smart-contracts-6-solidity-vulnerabilities-and-how-to-avoid-them-part-1-c33048d4d17d
  - Unprotected `selfdestruct` https://articles.caster.io/blockchain/self-destructing-smart-contracts-in-ethereum/
  - Older versions of `solc`, deprecated constructions https://github.com/runtimeverification/verified-smart-contracts/wiki/List-of-Security-Vulnerabilities
  - Integer overflows/underflows https://medium.com/loom-network/how-to-secure-your-smart-contracts-6-solidity-vulnerabilities-and-how-to-avoid-them-part-1-c33048d4d17d

Study of Security Vulnerabilities

Security Vulnerabilities - Re-entrancy

- A re-entrancy condition is when an actor can call your contract's function and potentially *re-enter* the contract before the previous call is completed - one or multiple times.
  - This would be especially disastrous in the case of a payable function.
  - The fact that the `call()` function triggers code execution without setting a gas limit makes in vulnerable to re-entrancy bugs.
  - The **DAO Attack** was a re-entrancy attack.

```
mapping (address => uint) private userBalances;

function withdrawBalance() public {
    uint amountToWithdraw = userBalances[msg.sender];
    require(msg.sender.call.value(amountToWithdraw)());
    userBalances[msg.sender] = 0;
}
```

Security Vulnerabilities - Function Visibility

- Solidity has four visibility modifiers - public, private, external and internal.
- By default Solidity functions are set to **public**, therefore it is recommended to always explicitly mention the visibility modifiers.
- It has been observed that these attacks are quite **frequent** (and **dangerous**) because of improper constructor naming - either due to a **typographical error** or improper **copy-pasting** of existing contracts.
- The **Parity bug** was a vulnerability of this category.

```
contract HashForEther {
function withdrawWinnings() {
// Winner if the last 8 hex characters of the address are 0.
        require(uint32(msg.sender) == 0);
        _sendWinnings();
        }

function _sendWinnings() {
        msg.sender.transfer(this.balance);
        }
}
```

Security Vulnerabilities - Improper Access Control

- **Use of `tx.origin` for authorization**
  - A type of a phishing attack
  - If `tx.origin` is used for authorization, and the actual owner is conned to call a malicious contract which in turn calls the victim contract, then the authorization fails.
- **Forced Ether Reception**
  - If ether is forced using `selfdestruct` the receiving function's fallback function does not get executed.
  - Therefore, the contract's balance should never be used as a guard

```
contract Phishable {
            address public owner;
            constructor (address _owner) {
                        owner = _owner;
            }
function () public payable { } // collect ether
function

function withdrawAll(address _recipient) public {
            require(tx.origin == owner); _
            recipient.transfer(this.balance);
            }
            }
```

```
import "Phishable.sol";
contract AttackContract {
            Phishable phishableContract;
            address attacker; // The attackers address to receive  funds.
            constructor (Phishable _phishableContract, address
            _attackerAddress) {
            phishableContract = _phishableContract;
            attacker = _attackerAddress;
}
function () payable {
             phishableContract.withdrawAll(attacker);
            }
}
```

Security Vulnerabilities - Front Running

- Two transactions can be sent to the transaction pool and the **order** in which they arrive is irrelevant.
- **Gas** sent is vital as it usually decides which transaction is mined first.
- An attacker can also be a **miner** and miners decide the order of transactions.
- Creates a problem for smart contracts that rely on **state of storage variables.**

```
contract TransactionOrdering {
        uint256 price;
        address owner;

        function buy() returns (uint256) {
        Purchase(msg.sender, price);
        return price;
    }

        function      setPrice(uint256      _price)
ownerOnly() {
        price = _price;
        PriceChange(owner, price);
        }
}
```

Security Vulnerabilities - Overflows and Underflows

- An overflow is when a number gets incremented above its maximum value. Solidity can handle up to **256 bit numbers** (up to 2^{256} - 1), so incrementing by 1 would result in 0.
- Likewise, in the inverse case, when the number is unsigned, decrementing will underflow the number, resulting in the maximum possible value.

```
contract OverflowUnderFlow {
    uint public zero = 0;
    uint public max = 2**256-1;

    // zero will end up at 2**256-1
    function underflow() public {
        zero -= 1;
    }
    // max will end up at 0
    function overflow() public {
        max += 1;
    }
}
```

Security Vulnerabilities - Bad Randomness

- Many smart contracts, such as those made to create lotteries or online games, use **pseudo-random number generator** where the initialization seed is chosen uniquely for all miners.
- A common choice for the seed is the **hash** or **timestamp** of some block that will appear on the blockchain in the future. This ensures that at run-time the value is same for everyone and this is secure since content of future blocks is unpredictable
- However, **malicious miners** can craft their attacks to bias the seed

A New Taxonomy for the Vulnerabilities

Solidity	Re-entrancy	
	Access Control	Protection Issues
		Authorization through `tx.origin`
		Unprotected Ether Withdrawal
		Unprotected `selfdestruct`
		Unexpected Ether
		Visibility Issues
		Function Visibility
		Variable Visibility
	Arithmetic Issues	Integer Overflow & Underflow
		Floating Point & Precision
	Solidity Programming Issues	Uninitialized Storage Pointers
		Variable Shadowing
		Keeping Secrets
		Type Casts
		Lack of Proper Signature Verification
		Write to Arbitrary Storage Location
		Incorrect Inheritance Order
		Typographical Errors
		Use of Assembly
		Use of Deprecated Functions/Constructions
		Floating or No Pragma
		Outdated Compiler Version
	Exception Handling	Unchecked Call
		Gasless Send
		Call Stack Limit
		Assert Violation
		Requirement Violation
	Call to the Unknown	Dangerous Delegate Call
		External Contract Referencing
	Denial of Service	DoS with block gas limit
		DoS with failed call
EVM	Short Address Attack	
	Immutable bugs	
	Stack size limit	
Blockchain	Bad Randomness	
	Untrustworthy Data Feeds	
	Transaction Order Dependence	
	Timestamp Dependence	
	Unpredictable state (Dynamic Libraries)	

Vulnerability Benchmark

Vulnerability Benchmark Creation

- We collected contracts known to be vulnerable from various sources. This included -
  - Smart Contract Weakness Classification (SWC) Registry
  - (Not So) Smart Contracts
  - EVM Analyzer Benchmark Suite
  - Research papers, theses and books
  - Various blog posts, articles, etc.

Vulnerability Benchmark Creation

- **The Ponzi Scheme Menace**
  - It has been observed that the Ethereum Smart Contracts have become a breeding ground for ponzi schemes over the last few years.
  - Some of these Ponzi Schemes utilize the vulnerabilities explained earlier.
  - The others can be categorized into -
    - **Does Not Refund**, and
    - **Allows owner to withdraw funds**

Contract Name	Vulnerability	Address
DynamicPyramid	Does not Refund	0xa9e4E3b1DA2462752AeA980698c335E70E9AB26C
GreedPit	Does not Refund; Allow Owner to withdraw funds	0x446D1696a5527018453cdA3d67aa4C2cd189b9f6
NanoPyramid	Does not Refund	0xe19e5f100d6a31169b5dcA265C9285059C41D4F6
Tomeka	Does not Refund	0x24Ec083b6A022099003e3D035fed48b9a58296E5
ProtectTheCastle	Allow Owner to withdraw funds	0x7D56485e026D5D3881F778E99969D2b1F90c50aF
EthVentures	Allow Owner to withdraw funds	0x99D982E49bCB5465A6B4c1e0eC4341c912D9Ba42

The Vulnerability Benchmark - On-chain Contracts

Contract Name	Vulnerability	Address
SmartBillions	Bad Randomness	0x5acE17f87c7391E5792a7683069A8025B83bbd85
Lottery	Bad Randomness	0x80ddae5251047d6CeB29765f38FED1C0013004b7
EtherLotto	Bad Randomness	0xA11E4ed59dC94e69612f3111942626Ed513cB172
Ethraffle_v4b	Bad Randomness	0xcC88937F325d1C6B97da0AFDbb4cA542EFA70870
BlackJack	Bad Randomness	0xA65D59708838581520511d98fB8b5d1F76A96cad
LuckyDoubler	Bad Randomness	0xF767fCA8e65d03fE16D4e38810f5E5376c3372A8
EthStick	Bad Randomness, Floating Point Precision	0xbA6284cA128d72B25f1353FadD06Aa145D9095Af
TheRun	Bad Randomness, Transaction Order Dependence	0xcac337492149bDB66b088bf5914beDfBf78cCC18
Parity MultiSig Wallet	Dangerous Delegate Call, Denial of Service, Function Default Visibility	0x863DF6BFa4469f3ead0bE8f9F2AAE51c91A907b4
GovernMental Ponzi Scheme	Denial of Service, TimeStamp Dependence	0xF45717552f12Ef7cb65e95476F217Ea008167Ae3
Private_Bank	External Contract Referencing	0x95D34980095380851902ccd9A1Fb4C813C2cb639
StackyGame	Function Default Visibility	0x8f13a1d43408b6434dd10e161361386f3952d665
GoodFellas	Function Default Visibility	0x5E84C1A6E8b7cD42041004De5cD911d537C5C007
Rubixi	Function Default Visibility, Immutable Bugs	0xe82719202e5965Cf5D9B6673B7503a3b92DE20be
BeautyChain (BEC)	Integer Overflow	0xC5d105E63711398aF9bbff092d4B6769C82F793D
MESH	Integer Overflow	0x3AC6cb00f5a44712022a51fbace4C7497F56eE31
UGToken	Integer Overflow	0x43eE79e379e7b78D871100ed696e803E7893b644
SMT	Integer Overflow	0x55F93985431Fc9304077687a35A1BA103dC1e081
SMART	Integer Overflow	0x60be37dacb94748a12208a7ff298f6112365e31f
MTC	Integer Overflow	0x8febf7551eea6ce499f96537ae0e2075c5a7301a
GGToken	Integer Overflow	0xf20b76ed9d5467fdcdc1444455e303257d2827c7
CNYToken	Integer Overflow	0x041b3eb05560ba2670def3cc5eec2aeef8e5d14b
CNYTokenPlus	Integer Overflow	0xfbb7b2295ab9f987a9f7bd5ba6c9de8ee762deb8
DAO	Reentrancy	0xBB9bc244D798123fDe783fCc1C72d3Bb8C189413
SpankChain	Reentrancy	0xf91546835f756DA0c10cFa0CDA95b15577b84aA7
CityMayor	Reentrancy	0x4bdDe1E9fbaeF2579dD63E2AbbF0BE445ab93F10
ICO	Transaction Order Dependance	0xd80cc3550Da18313aF09fbd35571084913Cd5246
LastIsMe	Transaction Order Dependance	0x5D9B8FA00C16BCafaE47Deed872E919C8F6535BF
FirePonzi	Typographical error	0x062524205cA7eCf27F4A851eDeC93C7aD72f427b
KingofTheEtherThrone	Unchecked External Call	0xb336a86e2feb1e87a328fcb7dd4d04de3df254d0
EtherPot	Unchecked External Call	0x539f2912831125c9B86451420Bc0D37b219587f9
OpenAddressLottery	Unitialized Storage Pointers	0x741F1923974464eFd0Aa70e77800BA5d9ed18902
CryptoRoulette	Unitialized Storage Pointers	0x8685631276cFCf17a973d92f6DC11645E5158c0c
G_GAME	Unitialized Storage Pointers	0x3CAF97B4D97276d75185aaF1DCf3A2A8755AFe27

The Vulnerability Benchmark

- The final benchmark contains **180** contracts (162 are unique). This also contains **40** on-chain vulnerable contracts.

Effectiveness of the Security Tools

Security Tools

Security Tools

	Remix IDE	Smart-Check	Slither	Oyente	Securify	Mythril
Version/ Date Used	4-Mar-2019	2.0.1	0.4.0	0.2.7	17-Apr-19	0.20.4
Technique	Static Analysis	Static Analysis	Static Analysis	Symbolic Execution	Symbolic Execution	Symbolic Execution
WUI/ CLI	WUI	WUI + CLI	CLI	WUI + CLI	WUI + CLI	WUI + CLI
Works on src-file/ bytecode	src-file	src-file	src-file	src-file + bytecode	src-file + bytecode	src-file + bytecode
Developed by	Ethereum Foundation	SmartDec	Trail of Bits	NUS + Melonport	ETH Zurich	ConsenSys

The Experimental Setup

- All the experiments were carried out on a machine running Ubuntu 18.04.2 LTS on an Intel® Core™ i7-4770 CPU with 16GB DDR3 RAM.

Security Tools - Claimed Vulnerability Coverage

		Remix	Slither	SmartCheck	Oyente	Mythril	Securify	SUM
SOLIDITY	Re-entrancy	Y	Y		Y		Y	4
	Authorization through `tx.origin`	Y	Y			Y		3
	Unprotected Ether Withdrawal		Y			Y	Y	3
	Unprotected `selfdestruct`	Y	Y			Y		3
	Unexpected Ether		Y	Y				2
	Function Visibility			Y				1
	Variable Visibility							0
	Integer Overflow & Underflow				Y	Y		2
	Floating Point & Precision			Y				1
	Uninitialized Storage Pointers		Y					1
	Variable Shadowing		Y					1
	Keeping Secrets			Y				1
	Type Casts						Y	1
	Lack of Proper Signature Verification							0
	Write to Arbitrary Storage Location			Y			Y	2
	Incorrect Inheritance Order							0
	Typographical Errors							0
	Use of Assembly	Y	Y	Y				3
	Use of Deprecated Functions		Y	Y		Y		3
	Floating or No Pragma			Y				1
	Outdated Compiler Version		Y	Y				2
	Unchecked Call	Y	Y	Y			Y	4
	Gasless Send						Y	1
	Call Stack Limit						Y	1
	Assert Violation						Y	1
	Requirement Violation						Y	1
	Dangerous Delegate Call		Y			Y	Y	3
	External Contract Referencing							0
	DoS with block gas limit			Y			Y	2
	DoS with failed call		Y	Y			Y	3
EVM	Short Address Attack							0
	Immutable bugs							0
	Stack size limit				Y			1
B/CHAIN	Bad Randomness	Y		Y		Y		3
	Untrustworthy Data Feeds							0
	Transaction Order Dependence				Y		Y	2
	Timestamp Dependence	Y	Y	Y	Y			4
	Unpredictable state							0
PS	Does not Return		Y	Y			Y	3
	Allows Owner to Withdraw Funds							0
	TOTAL	7	15	15	5	7	14	

Results on the Benchmark

Results on the Benchmark

Tool-wise Analysis

Remix IDE

- **Manual** Static Analysis & **Incompleteness**
- For **`solc`<=0.3.1**, the `check-effects` and the `selfdestruct` modules give an error

Tool-wise Analysis

Slither

- Detected **at least one** from each category it claims to identify and was able to analyze successfully
- **Drawback** - requires correct `solc` version installed and does not work prior to 0.4.0

Tool-wise Analysis

SmartCheck

- Performance not very good
- Can detect simpler vulnerabilities effectively

Tool-wise Analysis

Oyente

- Showing its age
- Average EVM code coverage for the entire benchmark set was found to be **75.98%**.
- Not even a single case of integer overflow/underflow - bug in the tool

Tool-wise Analysis

Mythril

- It is able to detect the attacks with a fair accuracy, however it encounters a lot of errors.

Tool-wise Analysis

Securify

- The only tool that reports a contract as ‘safe’ from a particular vulnerability.
- Lot of false negatives reported by the tool

Malicious Nodes in Ethereum

Motivation

- Permissionless blockchains mainly deal with crypto-currencies and are prone to cyber attacks, scams and ransom payments.
- Can we detect such activities and generate alerts?

Motivation

- Blockchain Transaction graph is a temporal graph
  - Current existing techniques use aggregated snapshot to perform analysis of malicious activity and thus neglect temporal aspects such as behaviour changes over time.
  - They use graph properties such as inDegree, outDegree and clustering coefficient on top of blockchain specific properties such as transaction amount.

Motivation

- We find that each account behaves differently
  - Distribution of inDegree over time follows power law
    - suggesting bursty behaviour
  - Distribution of inter-event time follows truncated power law
    - suggesting temporal burstiness
  - Most malicious accounts interact with accounts that they have not interacted in past (stability of neighbourhood is low).

inDegree burst

Low stability

Our Approach

Features Extraction

- Based on previous attacks, scams and ransom payments
- Extended existing feature set to include temporal behaviour based features related to
  - Burstiness
    - In/out-Degree
    - Temporal
    - Transaction Amount
  - Attractiveness

Burstiness: Degree

- More than k events happening at the same time.

- Attacks such as Allinvain theft uses such feature.

Burstiness: Temporal

- Events happening for at least m consecutive time instances

- Accounts involved in Gambling show such behaviour

Burstiness: Transaction Amount

- Transaction amount more than Θ
- Silk Road accounts showed such transaction amount patterns

Attractiveness

- Measure of stability of neighbourhood.
  - N_i^t neighbourhood of account i at time t
  - Θ_a = duration of dataset

$$A_i^t = \begin{cases} 1 - \frac{|N_i^t \cap (\cup_{j \in T-\{t\}} N_i^j)|}{|\cup_{j \in T} N_i^j|}, & \text{when } t \geq \theta_a \text{ and } N_i^t \neq \emptyset \\ 0, & \text{otherwise.} \end{cases}$$

Supervised learning

- Using AutoML tool called TPOT
  - Configured to techniques used in related work and more.

- Extra Tree Classifier performs best with respect to balanced accuracy

Features	Data Segment	TPOT identified Classifier	Accuracy balanced	Precision		Recall		F1 score	
				M	B	M	B	M	B
25 (PCA)	Only EOA	ExtraTree	0.863	0.29	1.00	0.74	0.99	0.41	1.00
	EOA and SC	ExtraTree	0.884	0.21	1.00	0.77	0.99	0.34	0.99
34	Only EOA	ExtraTree	0.891	0.20	1.00	0.82	0.99	0.32	0.99
	EOA and SC	RF	0.895	0.65	1.00	0.80	1.00	0.71	1.00
41	Only EOA	ExtraTree	0.887	0.19	1.00	0.81	0.99	0.31	0.99
	EOA and SC	ExtraTree	0.892	0.21	1.00	0.80	0.99	0.33	0.99

[25 (PCA) EOA] ExtraTreesClassifier(class_weight = 'balanced', criterion = 'entropy', max_features = 0.55, max_samples = 0.1, min_samples_leaf = 16, min_samples_split = 15, n_estimators = 200)

[25 (PCA) EOA and SC] ExtraTreesClassifier(class_weight = 'balanced', criterion = 'entropy', max_features = 0.45, max_samples = 0.3, min_samples_leaf = 19, min_samples_split = 17, n_estimators = 200, random_state = 100)

[34 EOA] ExtraTreesClassifier(class_weight = 'balanced', criterion = 'entropy', max_features = 0.25, max_samples = 0.3, min_samples_leaf = 14, min_samples_split = 12, n_estimators = 400, random_state = 100)

[34 EOA and SC] RandomForestClassifier(bootstrap = False, class_weight = 'balanced', max_features = 0.35, max_samples = 0.65, min_samples_leaf = 19, min_samples_split = 12, n_estimators = 400, random_state = 100)

[41 EOA] ExtraTreesClassifier(max_features = 0.25, max_samples = 0.1, min_samples_leaf = 19, class_weight = 'balanced', min_samples_split = 20, n_estimators = 200, random_state = 100)

[41 EOA and SC] ExtraTreesClassifier(class_weight = 'balanced', criterion = 'entropy', max_features = 0.3, max_samples = 0.6, n_estimators = 600, min_samples_leaf = 19, min_samples_split = 14)

Table 2: Balanced accuracy, Precision, Recall and F1 score for both malicious (M) and benign (B) accounts with best identified ML algorithm for supervised case when using different dataset configurations. Here, RF represents RandomForest Classifier.

Unsupervised learning

- K-Means performed best and identified 19 clusters for one configuration (21 for another).
- We chose the cluster with most malicious nodes among identified clusters to identify behavioural similarity between accounts in the cluster.

Related Work

TABLE I: Features used in related studies

		Features based on												
#	Blockchain	Activeness	InDegree	outDegree	Balance	Gas/Transaction Fee	Bursty Behavior	Attractiveness	Clustering Coeff	Inter-event Time	ML Algo Used	Dataset	Hyperparameter	Performance
[11]	B	✓	✓	✓	✓	-	-	-	✓	✓	K-means	$100K^a$	$k \in [1, 14]$	$k_{opt} = 7, 8$
											Mahalanobis Distance		$\times$	0.0256^{MDE}
											ν-SVM		$\nu = 0.005$	0.1441^{MDE}
[12]	B	✓	✓	✓	✓	-	-	-	-	✓	local outlier factor	$6.3M^a$	$k = 8$	0.55^{MDE}
[13]	B	-	✓	✓	✓	-	-	-	✓	-	K-means	$1M^a$	$k \in [1, 14]$	$k_{opt} = 8$
											Trimmed K-means		$k \in [1, 15]$, $\alpha = 0.01$	$k_{opt} = 8$
[14]	B	✓	✓	✓	✓	-	-	-	-	✓	RIPPER†	$\ddagger 6432^a$	cost $\in [1, 40]$	0.996^{ac}
											Bayes Network		$\times$	0.983^{ac}
											Random Forest		$\times$	0.996^{ac}
[15]	E	-	✓	✓	✓	✓	-	-	-	-	XGBoost	$\ddagger 1382^{sc}$	$\times$	0.94^p, 0.81^r
[16]	E	✓	✓	✓	-	✓	-	-	-	-	Random Forest	$350K^a$	RFPARAM	0.85^r
											SVM		$cost = 1$, $\gamma = 0.077$	0.87^r
											XGBoost		XGBPARAM	0.8^r
[17]	E	✓	✓	✓	✓	-	-	-	-	-	Decision Tree	300^a	$\times$	0.93^{ac}
											SVM		$\times$	0.83^{ac}
											KNN		$k = 5$	0.91^{ac}
											MLP		$\times$	0.86^{ac}
											NaiveBayes		$\times$	0.89^{ac}
											Random Forest		$\times$	0.99^{ac}
[18]	B	-	✓	✓	✓	✓	-	-	-	-	Adaboost	$1000M^a$	$estimators = 50$, $rate = 1$	$> 0.2^r$
											Random Forest		$estimators = 10$	$> 0.85^r$
											Gradient boosting		$estimators = 100$, $rate = 0.1$ $depth = 3$	$> 0.93^r$

^{B} Bitcoin, ^{E} Ethereum, ^{a} accounts, ^{sc} smart contracts, ^{MDE} Dual Evaluation Metric, ^{ac} accuracy, ^{p} Precision, ^{r} Recall, † it is a propositional rule learner that relies on a sequential covering logic, ‡ ponzi scheme data, ^{RFPARM} features = 3, leaf samples = 10, threshold probability = 0.99, ^{XGBPARAM} depth = 3, child weight = 8, subsample = 1, probability = 0.99, $^\times$ not provided.

Blockchain Technology And Applications

IIT Kanpur

C3I Center

Acknowledgement

- IBM Hyperledger presentations/slides
- Slides of Praveen Jayachandran, IBM

Key Concepts and Benefits of Blockchain for Business

Append-only distributed system of record shared across business network	**Shared Ledger`**	**Security**	Ensuring appropriate visibility; transactions are secure, authenticated & verifiable
Business terms embedded in transaction database & executed with transactions	**Smart Contracts**	**Consensus**	All parties agree to network verified transaction

Reduces Time	Removes Cost	Reduces Risk	Enables New Business Models
Transaction time from days to near instantaneous	Overheads and cost intermediaries	Tampering, fraud & cyber crime	IoT Integration into supply chain

Degree of Centralisation

Censorship-resistant **Privacy**

Scale to large number of nodes **Scale in transaction throughput**

One global blockchain **Many interacting blockchains**

Figure source: "Distributed Ledger Technology: Beyond Blockchain", A report by UK Govt Chief Scientific Adviser

The Linux Foundation Hyperledger Project

A collaborative effort created to advance blockchain technology by identifying and addressing important features for a cross-industry open standard for distributed ledgers that can transform the way business transactions are conducted globally.

www.hyperledger.org

Premier

General

Associate

108+ Members, 260% Growth in 11 months

Skry

International Trade & Supply Chain: Use Cases and Client Examples

Workflow Automation & Compliance

Automate current inefficient, manual and error-prone workflows in documentary trade finance

Supply-chain Visibility

Provide single view for purchase order life-cycle across the supply-chain as the *truth*

Trade/Supply-Chain Finance

Improve the efficiency of commercial financing business by sharing data in a secure and transparent manner

Supply-chain Provenance

Provide provenance across the supply-chain cutting through complex distribution and processing ecosystems

Trade Logistics	Invoice discounting	Trade finance	Food Safety	Channel Financing
MAERSK	Mahindra Rise.	HSBC, Bank of America Merrill Lynch	Walmart	IBM Global Financing

The Participants in a Blockchain Network

Blockchain Components

Blockchain

Ledger		contains the current world state of the ledger and a Blockchain of transaction invocations
Smart Contract	f(abc);	encapsulates business network transactions in code. transaction invocations result in gets and sets of ledger state
Consensus Network		a collection of network data and processing peers forming a Blockchain network. Responsible for maintaining a consistently replicated ledger
Membership	E T	manages identity and transaction certificates, as well as other aspects of permissioned access
Events		creates notifications of significant operations on the Blockchain (e.g. a new block), as well as notifications related to smart contracts. Does not include event distribution.
Systems Management	i	provides the ability to create, change and monitor Blockchain components
Wallet		securely manages a user's security credentials
Systems Integration		responsible for integrating Blockchain bi-directionally with external systems. Not part of Blockchain, but used with it.

Blockchain Applications and the Ledger

* Smart Contract implemented using chain code

Hyperledger Fabric Model

- **Permissioned** system; strong **identity management**
- Distinct roles of **users**, and **validators**
- Users **deploy** new pieces of code (chaincodes) and **invoke** them through **deploy** & **invoke** transactions
- Validators evaluate the effect of a transaction and reach consensus over the new version of the **ledger**
- **Ledger** = total order of transactions + hash (global state)
- **Pluggable consensus** protocol, currently PBFT & Sieve

Security & privacy features

Privacy of user-participation	Each user has control over the degree to which its transaction activity will be shared with its environment
Contract Privacy	Contract logic can be confidential, i.e., concealable to unauthorized entities
Accountability Non-repudiation	Users can be accounted for the transactions they create, cannot frame other users for their transactions, or forge other users' transactions.
Auditability	Auditors are able to access & verify any transaction they are legally authorized to

A sample transaction (1/6)

1. The Client App proposes a transaction for **Smart Contract A** to the Endorsing peer E_0. Endorsement policy: "E_0, E_1 and E_2 must sign". E_3 is not part of the policy

A sample transaction (2/6)

2. Endorsing peer E_0 endorses a tx and (optionally) "anchors it" with respect to the ledger state version numbers. An "anchor" contains all data read and written by contract that are to be confirmed by other endorsers.

A sample transaction (3/6)

3. The client requests further endorsement from E_1 and E_2.The client may decide to suggest an anchor obtained from E_0. to E_1 and E_2.

A sample transaction (4/6)

4. The Endorsing peers E_1 and E_2 send the endorsement to client.

A sample transaction (5/6)

5. Client formats the transaction and broadcasts it to the consenters for inclusion in the ledger

A sample transaction (6/6)

6. The consensus service delivers the next block in the ledger with the consented transaction.

Blockchain Technology And Applications

Sandeep K. Shukla

IIT Kanpur

C3I Center

Acknowledgement

- Fred Schneider (Cornell Univ)
- Elli Androulaki et. al (Authors of Hyperledger Fabric paper)
  https://arxiv.org/ct?url=https%3A%2F%2Fdx.doi.org%2F10.1145%2F3190508.3190538&v=92886ca2

Evolution of Blockchain Technology

- 1st generation: Store and transfer of value (e.g. Bitcoin, Ripple, Dash)
- 2nd generation: Programmable via smart contracts (E.g. Ethereum)
- 3rd generation: Enterprise blockchains (E.g. Hyperledger, R3 Corda & Ethereum Quorum)
- Next gen: Highly scalable with high concurrency

Source: SmartBridge: Glenn Jones, Ken Staker

Order-execute Paradigm

- Many existing smart-contract blockchains follow the blueprint of SMR (state-machine-replication) and implement so-called active replication:
  - a protocol for consensus or atomic broadcast first orders the transactions and propagates them to all peers
  - each peer executes the transactions sequentially
  - order-execute architecture
  - it requires all peers to execute every transaction and all transactions to be deterministic.
- The order-execute architecture can be found in most permission-less existing blockchain systems, and even in most permissioned ones
  - public ones such as Ethereum (with PoW-based consensus)
  - Permissioned ones (with BFT-type consensus) such as Tendermint, Chain and Quorum
- every peer executes every transaction and transactions must be deterministic

Other issues with other Blockchains

- Consensus is hard-coded within the platform, which contradicts
  - that there is no “one-size-fits-all” (BFT) consensus protocol
- The trust model of transaction validation is determined by the consensus protocol
  - cannot be adapted to the requirements of the smart contract
- Smart contracts must be written in a fixed, non-standard, or domain-specific language,
  - which hinders wide-spread adoption and may lead to programming errors
- The sequential execution of all transactions by all peers limits performance, and complex measures are needed to prevent denial-of-service attacks against the platform originating from untrusted contracts
  - Such as accounting for runtime with “gas” in Ethereum
- Transactions must be deterministic, which can be difficult to ensure programmatically
- Every smart contract runs on all peers,
  - which is at odds with confidentiality, and prohibits the dissemination of contract code and state to a subset of peers.

Hyperledger Fabric

- Fabric is the first blockchain system to support the execution of distributed applications written in standard programming languages,
  - allows them to be executed consistently across many nodes, giving impression of execution on a single globally-distributed blockchain computer
- The architecture of Fabric follows a novel ***execute-order-validate*** paradigm for distributed execution of untrusted code in an untrusted environment.
- It separates the transaction flow into three steps, which may be run on different entities in the system:
  - executing a transaction and checking its correctness, thereby endorsing it (corresponding to "transaction validation" in other blockchains);
  - ordering through a consensus protocol, irrespective of transaction semantics
  - transaction validation per application specific trust assumptions, which also prevents race conditions due to concurrency.

Order-Execute Paradigm of Most Blockchains

Figure 1: Order-execute architecture in replicated services.

Order-Execute in PoW blockchain

- PoW-based permission-less blockchain such as Ethereum combines consensus and execution of transactions as follows:
  - (1) every peer (i.e., a node that participates in consensus) assembles a block containing valid transactions
    - (to establish validity, this peer already pre-executes those transactions)
  - (2) the peer tries to solve a PoW puzzle
  - (3) if the peer is lucky and solves the puzzle, it disseminates the block to the network via a gossip protocol
  - (4) every peer receiving the block validates the solution to the puzzle and all transactions in the block
- Effectively, every peer repeats the execution of the lucky peer from its first step.
- All peers execute the transactions sequentially (within one block and across blocks)

Limitations of Order-Execute Paradigm

- **Sequential execution** Executing the transactions sequentially on all peers limits the effective throughput
  - In contrast to traditional SMR, the blockchain forms a universal computing engine and its payload applications might be deployed by an adversary. A denial-of-service (DoS) attack:
    - could simply introduce smart contracts that take a very long time to execute.
    - a smart contract that executes an infinite loop
  - To cope with this problem, public programmable blockchains with a cryptocurrency account for the execution cost
    - Ethereum Gas
    - However, blockchain with no native currency does not have this facility

Limitations of Order-Execute

- **Non-deterministic code**. Operations executed after consensus in SMR must be deterministic, otherwise the distributed ledger "forks" and violates the basic premise of a blockchain, that all peers hold the same state.
  - This is usually addressed by programming blockchains in domain-specific languages
    - e.g. Ethereum Solidity
  - Requires additional learning by the programmer.
- Writing smart contracts in a general-purpose language (e.g., Go, Java, C/C++) instead accelerates the adoption of blockchain solutions

Limitations of Order-execute

- **Confidentiality of execution:** Usually they run all smart contracts on all peers.
- Many intended use cases for permissioned blockchains require confidentiality,
  - i.e., that access to smart contract logic, transaction data, or ledger state can be restricted.
- cryptographic techniques, ranging from data encryption to advanced zero-knowledge proofs and verifiable computation can help to achieve confidentiality but
  - comes with a considerable overhead
- it suffices to propagate the same state to all peers instead of running the same code everywhere.
  - Execution of a smart contract can be restricted to a subset of the peers trusted for this task, that vouch for the results of the execution.

Limitations of Order-Execute

- **Fixed trust model:** Most permissioned blockchains rely on BFT replication protocols to establish consensus
  - Such protocols typically rely on a security assumption that among $n > 3f$ peers, up to f are tolerated to misbehave and exhibit Byzantine faults
- The same peers often execute the applications as well, under the same security assumption
  - even though one could actually restrict BFT execution to fewer peers
- such a quantitative trust assumption, irrespective of peers' roles in the system, may not match the trust required for smart contract execution
  - trust at the application level should not be fixed to trust at the protocol level.
- A general purpose blockchain should decouple these two assumptions and permit flexible trust models for applications.

Limits of Order-execute

- **Hard-coded consensus:** all blockchain systems, permissioned or not, came with a hard-coded consensus protocol
- one may want to replace BFT consensus with a protocol based on an alternative trust such as Paxos/Raft and ZooKeeper (Kafka)

SMR (State Machine Replication Model)

- Can represent ***deterministic*** distributed system as *Replicated State Machine*
- Each replica reaches the same conclusion about the system ***independently***

Motivation

Motivation

Motivation

- Need replication for fault tolerance
- What happens in these scenarios without replication?
  - Storage - Disk Failure
  - Webservice - Network failure
- Be able to reason about failure tolerance
  - How badly can things go wrong and have our system continue to function?

State Machines

- *c* is a Command
- *f* is a Transition Function

State Machine Replication (SMR)

- The *State Machine Approach* to a fault tolerant distributed system
- Keep around *N* copies of the state machine

State Machine Replication (SMR)

- The *State Machine Approach* to a fault tolerant distributed system
- Keep around *N* copies of the state machine

SMR Requirements

SMR Requirements

Great!

SMR Requirements

SMR Requirements

Great!

SMR Requirements

SMR Requirements

- Replicas need to ***agree*** on the which requests have been handled

SMR Requirements

SMR Requirements

SMR Requirements

SMR Requirements

SMR Requirements

SMR Requirements

SMR Requirements

- Replicas need to handle requests in the same ***order***

SMR

- All non faulty servers need:
  - Agreement
    - Every replica needs to accept the same set of requests
  - Order
    - All replicas process requests in the same relative order

Implementation

- Agreement
  - Someone proposes a request; if that person is nonfaulty all servers will accept that request
    - Strong and Dolev [1983] and Schneider [1984] for implementations
    - Client or Server can propose the request

SMR Implementation

SMR Implementation

Non-faulty Transmitter

Implementation

- Order
  - Assign unique ids to requests, process them in ascending order.
  - How do we assign unique ids in a distributed system?
  - How do we know when every replica has processed a given request?

SMR Requirements

SMR Requirements

Assign Total Ordering

Request	ID
r0	1
r1	2

SMR Requirements

Assign Total Ordering

Request	ID
r0	1
r1	2

SMR Requirements

Assign Total Ordering

Request	ID
r0	1
r1	2

SMR Requirements

Assign Total Ordering

Request	ID
r0	1
r1	2

r0 is now stable!

SMR Requirements

Assign Total Ordering

Request	ID
r0	1
r1	2

r0 is now stable!

r1 is now stable!

Implementation Client Generated IDs

- Order via Clocks (Client timestamps represent IDs)
  - Logical Clocks
  - Synchronized Clocks
- Ideas from [Lamport 1978]

Implementation Replica Generated IDs

- 2 Phase ID generation
  - Every Replica proposes a *candidate*
  - One candidate is chosen and agreed upon by all replicas

Replica ID Generation

Replica ID Generation

Req.	CUID	UID
r0	1.1	
r1	2.1	

Req.	CUID	UID
r0	1.2	
r1	2.2	

Req.	CUID	UID
r1	1.3	
r0	2.3	

Req.	CUID	UID
r1	1.4	
r0	2.4	

1) Propose Candidates

Replica ID Generation

Req.	CUID	UID
r0	*1.1*	*2.4*
r1	*2.1*	

Req.	CUID	UID
r0	*1.2*	*2.4*
r1	*2.2*	

Req.	CUID	UID
r1	*1.3*	
r0	*2.3*	*2.4*

Req.	CUID	UID
r1	*1.4*	
r0	*2.4*	*2.4*

2) Accept *r0*

Replica ID Generation

Req.	CUID	UID
r0	*1.1*	*2.4*
r1	*2.1*	*2.2*

Req.	CUID	UID
r0	*1.2*	*2.4*
r1	*2.2*	*2.2*

Req.	CUID	UID
r1	*1.3*	*2.2*
r0	*2.3*	*2.4*

Req.	CUID	UID
r1	*1.4*	*2.2*
r0	*2.4*	*2.4*

3) Accept *r1*

Replica ID Generation

Req.	CUID	UID
r1	2.1	2.2
r0	1.1	2.4

Req.	CUID	UID
r1	2.2	2.2
r0	1.2	2.4

Req.	CUID	UID
r1	1.3	2.2
r0	2.3	2.4

Req.	CUID	UID
r1	1.4	2.2
r0	2.4	2.4

r1 is now stable

Replica ID Generation

Req.	CUID	UID
r1	*2.1*	*2.2*
r0	*1.1*	*2.4*

Req.	CUID	UID
r1	*2.2*	*2.2*
r0	*1.2*	*2.4*

Req.	CUID	UID
r1	*1.3*	*2.2*
r0	*2.3*	*2.4*

Req.	CUID	UID
r1	*1.4*	*2.2*
r0	*2.4*	*2.4*

4) Apply *r1*

Replica ID Generation

Req.	CUID	UID
r1	2.1	2.2
r0	1.1	2.4

Req.	CUID	UID
r1	2.2	2.2
r0	1.2	2.4

Req.	CUID	UID
r1	1.3	2.2
r0	2.3	2.4

Req.	CUID	UID
r1	1.4	2.2
r0	2.4	2.4

5) Apply *r0*

Implementation Replica Generated IDs

- 2 Rules for Candidate Generation/Selection
  - Any new candidate ID must be > the id of any *accepted* request.
  - The ID selected from the candidate list must be >= each candidate
- In the paper these are written as:
  - If a request *r'* is seen by a replica sm_i after *r* has been accepted by sm_i then $uid(r) < cuid(sm_i, r')$
  - $cuid(sm_i, r) <= uid(r)$

Fault Tolerance

- Fail-Stop
  - A faulty server can be detected as faulty
- Byzantine
  - Faulty servers can do arbitrary, perhaps malicious things
- Crash Failures
  - Server can stop responding without notification (subset of Byzantine)

Fault Tolerance

- Fail-Stop
  - A faulty server can be detected as faulty
- Byzantine
  - Faulty servers can do arbitrary, perhaps malicious things
- Crash Failures
  - Server can stop responding without notification (subset of Byzantine)

Fail-Stop Tolerance

Fail-Stop Tolerance

Req.	CUID	UID
r0	1.1	

1) Propose Candidates….

Fail-Stop Tolerance

Req.	*CUID*	*UID*
r0	*1.1*	*1.1*

2) Accept *r0*

Fail-Stop Tolerance

Req.	*CUID*	*UID*
r0	*1.1*	*1.1*

2) Apply *r0*

Fail-Stop Tolerance

GAME OVER!!!

2) Apply *r0*

Fail-Stop Tolerance

- To tolerate *t* failures, need *t+1* servers.
- As long as 1 server remains, we're OK!
- Only need to participate in protocols with other *live* servers

Byzantine Tolerance

Byzantine Tolerance

Byzantine Tolerance

Byzantine Tolerance

Client trusts the majority =>
Need majority to participate in replication

Byzantine Tolerance

Who to trust?? 3 or 7?

Byzantine Tolerance

Byzantine Tolerance

Req.	CUID	UID
r0	1.1	

Req.	CUID	UID
r0	1.2	

Req.	CUID	UID
r0	???	???

Req.	CUID	UID
r0	1.4	

1) Propose Candidates

Byzantine Tolerance : a) No response

Req.	CUID	UID
r0	1.1	

Req.	CUID	UID
r0	1.2	

Req.	CUID	UID
r0	???	???

Req.	CUID	UID
r0	1.4	

a) Wait for majority candidates
Timeout long requests & notify others

Byzantine Tolerance
a) No response

Req.	CUID	UID
r0	1.1	1.4

Req.	CUID	UID
r0	1.2	1.4

Req.	CUID	UID
r0	???	???

Req.	CUID	UID
r0	1.4	1.4

a) *Accept r0*

Byzantine Tolerance: Small ID

Req.	CUID	UID
r0	1.1	

Req.	CUID	UID
r0	1.2	

Req.	CUID	UID
r0	-5	???

Req.	CUID	UID
r0	1.4	

1) Propose Candidates

Byzantine Tolerance: Small ID

uid = max(cuid(sm$_i$,r))
Ignore low candidates!

Req.	CUID	UID
r0	1.1	

Req.	CUID	UID
r0	1.2	

Req.	CUID	UID
r0	-5	???

Req.	CUID	UID
r0	1.4	

2) *Accept r0*

Byzantine Tolerance: Small ID

uid = max(cuid(sm$_i$,r))
Ignore low candidates!

Req.	CUID	UID
r0	1.1	1.4

Req.	CUID	UID
r0	1.2	1.4

Req.	CUID	UID
r0	-5	???

Req.	CUID	UID
r0	1.4	1.4

2) *Accept r0*

Byzantine Tolerance: Large ID

Req.	CUID	UID
r0	1.1	

Req.	CUID	UID
r0	1.2	

Req.	CUID	UID
r0	10	???

Req.	CUID	UID
r0	1.4	

1) Propose Candidates

Byzantine Tolerance: Large ID

Large numbers follow protocol!

Req.	CUID	UID
r0	1.1	

Req.	CUID	UID
r0	1.2	

Req.	CUID	UID
r0	**10**	???

Req.	CUID	UID
r0	1.4	

Byzantine Tolerance: Large ID

Large numbers follow protocol!

Req.	CUID	UID
r0	1.1	10

Req.	CUID	UID
r0	1.2	10

Req.	CUID	UID
r0	**10**	???

Req.	CUID	UID
r0	1.4	10

2) *Accept r0*

Fault Tolerance

- Byzantine Failures
  - To tolerate t failures, need $2t + 1$ servers
  - Protocols now involve votes
    - Can only trust server response if the majority of servers say the same thing
  - $t + 1$ servers need to participate in replication protocols

How Blockchains differ from SMRs

- not only one, but many distributed applications run concurrently
- applications may be deployed dynamically and by anyone
- the application code is untrusted, potentially even malicious

How Fabric's Execute-order-validate works

Figure 2: Execute-order-validate architecture of Fabric (*rw-set* means a readset and writeset as explained in Sec. 3.2).

Fabric Overview

- Fabric is a **distributed operating system** for permissioned blockchains that
  - executes distributed applications written in general purpose programming languages (e.g., Go, Java, Node.js)
  - securely tracks its execution history in an append-only replicated ledger data structure and has no cryptocurrency built in

Smart Contract in Fabric

- A smart contract, called **chaincode**, which is program code that implements the application logic and runs during the execution phase.
- The chaincode is the central part of a distributed application in Fabric and may be written by an untrusted developer.
- Special chaincodes exist for managing the blockchain system and maintaining parameters, collectively called **system chaincodes**

Endorsement Policy

- An **endorsement policy** that is evaluated in the validation phase.
- Endorsement policies cannot be chosen or modified by untrusted application developers
- An endorsement policy acts as a static library for transaction validation in Fabric, which can merely be parameterized by the chaincode.
- Only designated administrators may have a permission to modify endorsement policies through system management functions.
- A typical endorsement policy lets the chaincode specify the endorsers for a transaction in the form of a set of peers that are necessary for endorsement
- It uses a logical expression on sets, such as “three out of five” or “$(A \wedge B) \vee C$.” Custom endorsement policies may implement arbitrary logic

Fabric Transaction flow

Figure 4: Fabric high level transaction flow.

Summary of Hyperledger Fabric Transaction Flow

- A client sends transactions to the peers specified by the endorsement policy
- Each transaction is then executed by specific peers and its output is recorded; this step is also called endorsement (no state change)
- After endorsement, transactions enter the ordering phase,
  - consensus protocol to produce a totally ordered sequence of endorsed transactions grouped in blocks.
- These are broadcast to all peers, with the (optional) help of gossip
- Each peer then validates the state changes from endorsed transactions with respect to the endorsement policy and the consistency of the execution in the validation phase (states updated)
- All peers validate the transactions in the same order and validation is deterministic.
- Fabric introduces a novel hybrid replication paradigm in the Byzantine model,
  - which combines passive replication (the pre-consensus computation of state updates) and
  - active replication (the post-consensus validation of execution results and state changes).

Blockchain Technology And Applications

Sandeep K. Shukla

IIT Kanpur

C3I Center

Acknowledgements

- IOTA while paper https://iota.org/IOTA_Whitepaper.pdf
- IOTA blog https://blog.iota.org/the-tangle-an-illustrated-introduction-4d5eae6fe8d4
- https://public-rdsdavdrpd.now.sh/

Transactions, Confirmation And Consensus

Internet of Things
"50 billion connected devices in 2020"
-Cisco

Blockchain and IoT

IoT – Internet of Things

- Examples
  - Smart City
  - Smart Home
  - Smart Grid
  - Smart Transportation
- What is enabling information technology?
  - IoT components must talk to each other M2M to share information
  - Visibility of the State of the system or subsystem as a whole for autonomous decision making
  - Cloud based IoT ecosystem proposed by many companies
  - All IoT devices communicate to the cloud and get global state info from the cloud
  - Often communicate via cloud

Cloud + IoT

Single Shared Truth for Everyone

What’s the problem with Cloud + IoT

- Single point of failure
- Data Integrity and confidentiality at stake
- Cyber attack on the cloud
- Cloud infrastructure provider gets enormous power and data
- Can we decentralized this?
  - Blockchain anyone?

IoT + Blockchain

- Existing Blockchain (Bitcoin, Ethereum etc)
  - Scalability issues
  - PoW computational requirement
  - Centralization by powerful miners
  - Cost of transactions
  - All guarantees of integrity is probabilistic
  - Privacy requires a bit more thought
- IoTA foundation claims to have a solution
  - Replace Blockchain by Tangle
  - It borrows a lot of ideas from Blockchain
    - But not exactly a blockchain

Requirements of IoT

- Low Resource Consumption
- Widespread Interoperability
- Billions of Nano-transactions
- Data Integrity

The Tangle

A Blockchain **without the Blocks** and the **Chain**

Tangle

- No block – individual transactions are tangled together
- What is Tangling
  - Construct Directed Acyclic Graphs (DAGs) connecting transactions
- Self Regulating
- Very Scalable
- Still use PoW – but a long overhead PoW
  - Prevent spamming

What we get out of Tangle in place of Blockchain?

- CAP
  - Consistency
  - Availability
  - Partition-Tolerance
- No Fees
- Scalable
- Modular
- Lightweight
- Offline allowed
- Quantum Proof

Envisioned Use cases

- Complete M2M communication
  - Anything which has computational resource (Chip) can be leased by another machine autonomously
  - Devices can share resources by coordinating – bandwidth sharing for example
  - Supply Chain
  - Smart Grid to coordinate production of energy without human dispatching
  - On-demand API access
  - Sensor Data Selling and Data Market Place
  - ….

Towards Smart Decentralization

Dumb **Decentralization**

- "Dumb" devices
- No connectivity / sharing of data
- Human mediators

Smart **Centralization**

- Smart devices, dumb network
- Cloud as decision maker

Smart **Decentralization**

- Data Sharing
- Local Real-time Decision Making
- Smart adaptive and intelligent network

Tangle Initialization & Transaction Issuance

Issuing a Transaction

1. Bundling & Signing
2. Tip Selection
3. Validation
4. Proof-of-Work (PoW)
5. Publishing

https://public-rdsdavdrpd.now.sh/

Simulations

- Simulation by varying λ (transaction arrival rate – Poisson process)
  - https://public-rdsdavdrpd.now.sh/
- Simulation of unweighted random walk based tip selection
  - https://public-xnmzdqumwy.now.sh/
- Simulation of weighted random walk based tip selection
  - https://public-qnbiiqwyqj.now.sh/
- Simulation of Confirmation Confidence Computation
  - https://public-krwdbaytsx.now.sh/
  -

Initial Tangle State

Adding A Transaction

Another Transaction

New Tangle State

Confirmation Levels

Propagation Delay

Double Spend

Double Spend Resolution

Offline Tangle

Blockchain Technology And Applications

Sandeep K. Shukla

IIT Kanpur

C3I Center

Acknowledgements

- Richard Brown, R3
- Razi Rais, Microsoft
- Corda Whitepaper: https://docs.corda.net/_static/corda-introductory-whitepaper.pdf
- Corda Technical Whitepaper: https://docs.corda.net/_static/corda-technical-whitepaper.pdf

What is Corda? Is it a Blockchain?

- Corda has no unnecessary global sharing of data: only those parties with a legitimate need to know can see the data within an agreement
- Corda choreographs workflow between firms without a central controller
- Corda achieves consensus between firms at the level of individual deals, not the level of the system
- Corda's design directly enables regulatory and supervisory observer nodes
- Corda transactions are validated by parties to the transaction rather than a broader pool of unrelated validators
- Corda supports a variety of consensus mechanisms
- Corda records an explicit link between human-language legal prose documents and smart contract code
- Corda is built on industry-standard tools
- Corda has no native cryptocurrency

Blockchains are basically 5 interlocking services

- Consensus
- Validity
- Uniqueness (anti-double-spend)
- Immutability
- Authentication

Whether all or subset of these services are required is based on the business problem we are trying to solve.

Business Problem of Bitcoin

- No one can stop me from spending my money

Business Problem of Financial Institutions

"The financial industry is pretty much *defined* by the agreements that exist between its firms and these firms share a common problem: the agreement is typically recorded by *both* parties, in *different* systems and ***very large* amounts of cost are caused by the need to fix things when these different systems end up believing different things.**"

Imagine we had a system for recording and managing financial agreements that was *shared* across firms, that recorded the agreement consistently and identically, that was visible to the appropriate regulators and which was built on industry-standard tools, with a focus on interoperability and incremental deployment and which didn't leak confidential information to third parties. A system where one firm could look at its set of agreements with a counterpart and know for sure that:

"***What I see is what you see and we both know that we see the same thing and we both know that this is what has been reported to the regulator***"

What do these institutions need to agree on?

- Bank A and Bank B agree that Bank A owes 1M USD to Bank B, repayable via RTGS on demand.
- *This is a cash demand deposit*
- Bank A and Bank B agree that they are parties to a Credit Default Swap with the following characteristics
- *This is a derivative contract*
- Bank A and Bank B agree that Bank A is obliged to deliver 1000 units of BigCo Common Stock to Bank B in three days' time in exchange for a cash payment of 150k USD
- *This is a delivery-versus-payment agreement*
- … and so on…

Corda

- Differences with typical blockchain
  - Consensus occurs between parties to deals, not between all participants.
  - Corda lets users write their validation logic in industry-standard tools and we define who needs to be in agreement on a transaction's validity on a contract-by-contract basis.
  - Brewer's CAP Theorem
  - Data is not broadcast to every one – only to stakeholders who "need to know"

Evolution of Data Sharing among Financial Institutions

Bi-lateral Reconciling **Intermediary based Reconciling** **Global Logical Ledger**

Source: Corda Whitepaper

Principal Features of Corda (1)

- Recording and managing the evolution of financial agreements and other shared data between two or more identifiable parties in a way that is grounded in existing legal constructs and compatible with existing and emerging regulation
- Choreographing workflow between firms without a central controller.
- Supporting consensus between firms at the level of individual deals, not a global system.
- Supporting the inclusion of regulatory and supervisory observer nodes.

Principal features (2)

- Validating transactions solely between parties to the transaction.
- Supporting a variety of consensus mechanisms.
- Recording explicit links between human-language legal prose documents and smart contract code.
- Using industry-standard tools.
- Restricting access to the data within an agreement to only those explicitly entitled or logically privileged to it.

Concepts

- Global Ledger – but transactions and ledger entries are not globally visible
- State object – digital document which records the existence, content, and current state of an agreement between two or more parties
  - Shared only between parties with legitimate stake in the agreement
- Secure cryptographic hashes are used to identify parties and data
- Ledger is defined as a set of immutable state objects
- All parties in an agreement should be in consensus as to the state of an agreement as it evolves

State Object Representing a Cash Claim against a Commercial bank

Corda Ledger

The Corda Ledger

The ledger from each peer's point of view is the union of all intersections with other network peers

(some of which may be the empty set)

Numbered circles represent unique shared facts

ALICE = { 1 7 }

BOB = { 1 7 6 5 }

CARL = { 9 4 6 5 2 3 }

DEMI= { 2 3 8 }

ED= { 9 4 8 3 }

Corda Consensus

- In bitcoin we do a consensus over the state of the entire ledger
- In Ethereum we do a consensus over the state of the entire global VM
- In Corda there are consensus between only stake holders on the state of the agreement (state object)
- Tools to achieve consensus in Corda:
  - Smart Contract Logic ensures that state transitions are valid according to pre-agreed rules
  - Uniqueness and timestamping services are used to order transactions temporarily to eliminate conflicts
  - An orchestration framework simplifies the process of writing complex multi-step protocols between multiple different parties

Corda Transactions

- In Corda, updates are applied using transactions
  - Transactions consume existing state objects
  - Output new state objects
- Two aspects of consensus
  - Transaction validity: -- check all contract codes that executes the transactions runs successfully, all required signatures are valid, and any referenced transaction is valid
  - Transaction uniqueness: -- there exists no other transaction, over which a consensus was reached, and that consumes any of the same states as this transaction
- Parties can agree on transaction validity by independently running the same contract code and validation logic
- Consensus on Transaction uniqueness requires a predetermined observer

Uniqueness Consensus

- Corda has pluggable uniqueness service
  - Improves privacy, scalability, legal-system compatibility, and Algorithmic agility
- A single service may be composed of many mutually untrusting nodes coordinating via a Byzantine fault-tolerant algorithm or could be a single machine
- Uniqueness service does not check validity of transactions hence do not need to see full content of transactions -- privacy

Corda Workflow

State Object

States are immutable objects that represent (shared) facts such as a financial agreement or contract at a specific point in time

Transaction

Transactions consume input states and create output states.

The newly created output states replace the input states which are marked as historic.

Consensus

Parties reach consensus on the evolution of a shared fact. This is done by testing the validity (by way of contract code) and uniqueness (by way of the notary) of the transaction.

Flows

Flows are light-weight processes used to coordinate interactions required for peers to reach consensus about shared facts.

Corda node architecture

Corda Network

Corda Business Network

p by a
vill include:

ks, and our
erate..

Blockchain Technology and Applications

IIT Kanpur

Summary Blockchains*

*By Ras Dwivedi, IIT Kanpur

Blockchain 1.0

Block		Chain		Ledger
• Allows transaction to be recorded in a Block	+	• Cryptographically align block in chronological order	=	• And allow resulting ledger to be accessed by participants

Bitcoin Blockchain

1970 Merkle tree:

1992 Proof of work

2008 Bitcoin

Bitcoin blockchain:

- Proof of Work + Transaction Validation
- Bitcoin rewards as incentive for mining and Keeping them Honest
- Permission-less Blockchain
- Idea that Block chained by Hash could be public ledger
- **Idea that <50% attack could be sustained**

Bitcoin Mining Pools

Problem

- High transaction time- 10 min/block
- Limited size of block
- Wastage of energy
- Pseudo-anonymous

Algorand

Algorand: The Solution

- New Byzantine agreement using proof of stake
- Uses verifiable random function and cryptographic sortition.
- No Forks: due to explicit consensus
- No proof of work: which means scaling easy
- Proven security bounds, about % of malicious users who may disrupt majority of network for certain amount of time.
- Very low CPU load, can potentially run on modern mobile phones

Algorand features

- Sybil attack not possible
- Double spending (safety) not possible
- No Forking (safety)
- Safe from DDoS attacks
- Transaction Efficiency

Broad Ideas

- Weighted users: Poof of stake
- Consensus by Committee
- Randomly selected committee votes in every round
- Participant replacement: No private state after voting
- Voter cannot be DoSed
- Selection by sortition:
- Users privately find out whether they are members of committee of not
  - Others can verify
- Multi-step Byzantine agreement

Algorand Advantage

- Security
  - Verifiable Random function protects the committee members
- Speed: Byzantine Fault tolerant algorithm increases speed
- No forks: network never have divergent views of confirmed transaction, even in case of network partition
- Latency: confirm transaction in order of minutes (no need to wait after block accepted)
- Problem: Patented, not open source

Blockchain 2.0

«Traditional» contract

Smart contract

Program execution
(Only one interpretation)
Not reversible, autonomous

Execution

Software
program

Contract

Real time
Immutable

Definition

Audit
(Control)

Ethereum Blockchain

- Smaller block generation time
- Smart contract: Turing complete language
- **Gas:** A measure of computational cost
- **Ether:** another cryptocurrency
- Dynamic pricing of gas
- New idea: proof of stake, Proof of Authority ….
- Still under progress
- Research on doing computations outside of EVM

Problems

- High latency:
  - 15 Tps in Ethereum, visa : 2000 Tps
  - for bitcoin it may take ~2hrs to be sufficiently permanent
- Forking: different user may have different versions of the chain
- Network partition attack is possible
- Energy wastage
- Collusion: users may pool >50% mining power and steal currency
- Scalability : when number of users grow

Blockchain 3.0 : DLT

HYPERLEDGER FABRIC

HYPERLEDGER

HYPERLEDGER SAWTOOTH

IROHA

Hyperledger Fabric

- Started from IBM
- Currently owned by: The Linux Foundation
- Open Source-Apache License
- Flexibility
- No mining

Characteristic	Ethereum	Hyperledger Fabric
Description of platform	– Generic blockchain platform	– Modular blockchain platform
Governance	– Ethereum developers	– Linux Foundation
Mode of operation	– Permissionless, public or private[4]	– Permissioned, private
Consensus	– Mining based on proof-of-work (PoW) – Ledger level	– Broad understanding of consensus that allows multiple approaches – Transaction level
Smart contracts	– Smart contract code (e.g., Solidity)	– Smart contract code (e.g., Go, Java)
Currency	– Ether – Tokens via smart contract	– None – Currency and tokens via chaincode

Asset
Ledger
Chain-
code
Channel
Membe
r-ship
Services
Consensu
s
Hyperledge
r

Peers and Ledger

Blockchain

Channels

Organization

Endorsement policy

Describes condition by which a transaction can be endorsed

- A transaction can only be considered valid if it has been endorsed according to its policy
- Each Chaincode is associated with an Endorsement policy

Examples of policies

- AND(‘org1.member’,’Org2.member’,’Org3.member’)
- Or(‘Org1.member’,’Org2.member’)
- Or(‘Org1.member’,AND(‘Org2.member’,’Org3.member’))

Proposal

N	Blockchain Network	S	Chaincode
C	Channel	O	Orderer
P	Peer	L	Ledger
T1 P	Transaction T proposal P	T1 R2 E2	Transaction T1, response R2 endorsed with E2
T T C	Ledger transaction T1 flows on channel C	PA C	Principal PA (P1,P2) communicates via channel C.

Packaging

A1
T1 R1 E1,E2
A2
T2 R2 E1
A...
T... R... E...
B2
T1 T2
T3 T4
T6 T5
O1
P1
P2
T1 R1 E1,E2
T2 R1 E1
T... R... E...
C
N
N
Blockchain Network
P
Peer
B1
Block B1
O
Orderer
T1 R2a E2
Transaction T1, response R2a endorsed with E2
C
Channel
T1 B1
T2
T3
Block B1 contains transactions T1, T2, T3...
T1
T1
C
Ledger transaction T1 flows on channel C
PA
C
Principal PA (P1,P2) communicates via channel C.

Validation

N	Blockchain Network	P	Peer
C	Channel	O	Orderer
L	Ledger	B	Block B
L1 B0 B1	Ledger L1 has blockchain with blocks B0, B1	T1 T2 T3 B1	Block B1 contains transactions T1, T2, T3...
B1 C	Block B1 flows on channel C	PA C	Principal PA (P1, P2) communicates via channel C.

Pros

- Enterprise backing
- Open source: Apache license
- Modular architecture
- Private channels
- Smart contracts
- No PoW: Kafka Crash fault tolerant consensus

Cons

- Permissioned
- Identities of parties must be known
- No cryptocurrency, no notion of reward

Blockchain without block and without chains

DAG
No Transaction fee
IOTA
Scalable
No Mining

IOTA Blockchain

- Public Blockchain
- Genesis transaction creates maximum number of IOTA tokenS that could be ever circulated
- Transaction are called sites, unconfirmed one are called tips
- Sites are weighted in proportion to computational work needed
- Does not support smart contract

- Neighbours are mutual tethering
- Each node stores entire state
- Nodes propagates information to all its neighbours
- Every transaction refers to two past transactions
- Uses PoW to confirm two transaction
- MCMC based selection of nodes for confirmation

Pros

- Good for micro payment
- Quantum immune
- No hierarchy of nodes
- Public Blockchain without rewards
- Network becomes faster as it grows
- Offline transaction is possible
- Parallel vs Sequential consensus

Cons

- Still uses Proof of Work
- Use ternary number system
- Crypto is not verified or tested: violates “Do not roll on your own crypto”
- Cannot handle smart contracts

KSI Blockchain

Keyless infrastructure

Hashing

Aggregation

Publication

CALENDAR
t_0
t_1
t_2
t_{n-1}
t_n
6000
CORE
5000
AGGREGATOR
AGGREGATOR
4000
GATEWAY
GATEWAY
GATEWAY
GATEWAY
GATEWAY
3000
CLIENT
CLIENT
CLIENT
CLIENT
CLIENT
CLIENT
CLIENT
CLIENT
CLIENT
2000

FIG. 2

t_n
5001
CORE
5000
AGGR
4010-1
AGGR
4010-k
4000
REQ
GATE WAY
3010-1
GATE WAY
REQ
3010-2
REQ
GATE WAY
3010-m
GATE WAY
REQ
3000
CLIENT
REQ
2020
2018
2010-1
h(-,-, . . .)
2016
API
2014
2012
CLIENT
2010-n
2000

KSI: Pros

- Privacy
- Publication of the root hash guarantees immutability
- No mining: Highly scalable
  - 10^{12} registration per second
- Quantum immune
- Indefinite key expiry

KSI: Cons

- Private/permissioned
- Patented by Guardtime
- Smart contract not available
- Loss of token

Summary

BFT
Cryptographic sorition
Zk-Snarks
National Blockchain
Hard printing
Permissioned blockchain
Nash Equilibrium

Credits images

https://arstechnica.com/tech-policy/2017/11/bitcoin-rival-doubles-in-price-in-four-days-as-bitcoin-price-slumps/

https://kryptomoney.com/ethereum-explores-higher-levels-heads-800/

https://image.slidesharecdn.com/smart-contracts-150925125324-lva1-app6892/95/smart-contracts-5-638.jpg?cb=1443185644

https://image.slidesharecdn.com/smart-contracts-150925125324-lva1-app6892/95/smart-contracts-4-638.jpg?cb=1443185644

https://bitcoinexchangeguide.com/algorand/

https://www.pinterest.co.uk/pin/5066618308753012/

https://chrispacia.wordpress.com/2013/09/02/bitcoin-mining-explained-like-youre-five-part-2-mechanics/

https://medium.com/@lhartikk/a-blockchain-in-200-lines-of-code-963cc1cc0e54

https://medium.com/hashtaagco/blockchain-say-what-8e16ed29e543

https://patents.google.com/patent/US20170033932

https://medium.com/iota-et-tangle/presentation-iota-tangle-2d6e63e3a70

https://www.bitcoinbeginner.com/blog/what-is-iota/

https://medium.com/@philippsandner/comparison-of-ethereum-hyperledger-fabric-and-corda-21c1bb9442f6

https://www.cryptocompare.com/mining/guides/how-to-mine-zcash/

https://www.criptonoticias.com/aplicaciones/zcash-celebra-primer-aniversario-impulsando-privacidad-intercambios-atomicos/

https://slack.com/apps/A0P9ZU35Z-crypto-currency-coin-market-cap

https://www.wired.com/story/how-to-set-up-twitter-lists/

https://www.vectorstock.com/royalty-free-vector/crypto-currency-zcash-silver-symbol-vector-19123605

https://atozforex.com/news/zcash-price-long-term-forecast-2025/

https://www.dreamstime.com/stock-illustration-thank-you-text-illustration-social-icons-tablet-computer-mobile-cellphones-cyan-digital-world-map-background-image48170847

Hyperledger Documentation

Blockchain Technology And Applications

Sandeep K. Shukla

IIT Kanpur

C3I Center

Land Records on Hyperledger Fabric

Hyperledger Fabric Basics: Peers

- Endorsing peers
- Committing peers
- Orderer

World State, Ledger and Blockchain

Hyperledger Fabric Basics: Channels

Hyperledger Fabric Basics: Organization

Hyperledger Fabric Basics: Orderers

P1,P2	Peers part of Org1
P3,P4	Peers part of Org2
L1	Ledger
CC	Chaincode
CA	Certificate Authority
MSP	Membership service provider

Fabric Architecture with 2 organizations

Land Registration

Example of multiple channel architecture

Problem Statement

- Design a decentralized system to capture Land records records
- Integrity of data should be maintained
- Privacy: could be optional (Designer’s choice)
- Verifiability: Owner should be able to verify his claim of ownership
- Ecosystem: Buyer, seller, Registration Org., Land Record Org.

Stakeholders

- **Registration Organization**: To register land record transaction and collect stamp duties
- **Land Record Organization:** To maintain land records
- **Citizens**

Land Record Management

- Registry (Transaction) based
  - Each land transaction is registered at the Registration office
- Update
  - Based on registration
  - Land record office validates the transactions and update the records

Apply Registration
Pays Stamp Duties
Checks Registration
Collects Stamp Duties
Apply to update land record
Receives New Land Records Copy
Checks update application
Update Land Records

CC1
CC2
P1
L1
L2
Citizen
Endorser: RO
Chaincode: Registration
CC2
CC3
P2
L2
L3
Registration Office
Chaincode: LandRecord Endorser: Land Record Organization
CC1
CC3
P3
L1
L3
Land Record Office
Chaincode: UpdateLandRecord Endorser: RO and LRO

CC1
CC2
L1
P1
L2
Channel 2
CC2
CC3
L2
P2
L3
Channel 1
CC1
CC3
P3
L1
L3
Channel 3

Querying chaincode of other channels is allowed
Channel 2
CC1
CC2
L1
P1
L2
CC2
CC3
L2
P2
L3
Channel 1
CC1
CC3
P3
L1
L3
Channel 3

Assets

Land Records

- Land Id
- Owner
- Land Area
- Circle Rate
- Percentage land Holding
- Registry

Registry

- Buyer
- Seller
- Sell Value
- Stamp Duty
- Date of Transaction
- Witness
- Registration number

Update Application

- Registration number
- Buyer
- Seller
- Land ID
- Date of Transaction

Transactions
(some)

1. applyRegistration
2. validateRegistration
3. applyUpdate
4. approveChanges
5. updateLandRecord
6. fetchLandRecords
7. getLandOwner

Process View

Deployment

Citizen

- Orderer
- Peer
- CA
- Fabric Rest API

Registration Org.

- Orderer
- Peer
- CA
- Fabric Rest API

Land Record Org.

- Orderer
- Peer
- CA
- Fabric Rest API

Land Record Transfer using Corda Blockchain

State in Corda

- A *state* is an immutable object representing a fact known by one or more Corda nodes at a specific point in time. States can contain arbitrary data, allowing them to represent facts of any kind (e.g. stocks, bonds, loans, KYC data, identity information…)
- A Land Record title can thus be treated as a state object in Corda and the transfer of this title can be portrayed by the state sequence which represents the lifecycle of a shared fact(in the present case a land ownership document) over time

System Design

- We will have the following participants as Corda nodes:
  - Buyer
  - Seller
  - Government
- Government can create a new land record document with the specified owner
- Buyer can request to buy a specific land record document
- Seller can sell(transfer ownership) of the land record document

System Design - Assets

- Land Record
  - Land ID (number)
  - Owner
    - name (string)
    - ID (number)
  - Previous owners (list of owner)
  - Buying requests (list of potential buyers)

System Design - Workflows

- Flows automate the process of agreeing ledger updates
- We will have the following flows:
  - Create Land Record Flow
    - Participants: Buyer, Government
    - Can be initiated by Government only
  - Request Ownership Transfer Flow
    - Participants: Seller, Buyer, Government
    - Can be initiated by Buyer only
  - Transfer Ownership Flow
    - Participants: Seller, Buyer, Government
    - Can be initiated by Seller only

System Design - Contracts

- A **contract** takes a transaction as input, and states whether the transaction is considered valid based on the contract's rules. In our application the contract should check for the following
- Create Land Record Contract
  - Zero input state & One output state
  - Initiated by Government
  - Two signers namely, buyer and government
- Request Ownership Transfer Contract
  - One input state & One output state
  - Initiated by any buyer
  - Three signers namely, buyer, seller and government
- Transfer Ownership Contract
  - One input state & One output state
  - Initiated by current owner
  - Three signers namely, buyer, seller and government

System Design - Consensus & Notary

- Determining whether a proposed transaction is a valid ledger update involves reaching two types of consensus:
- Validity consensus - this is checked by each required signer before they sign the transaction
  - In present application the signers would be buyer, seller and the government nodes
- Uniqueness consensus - this is only checked by a notary service
  - Notary clusters can have several nodes that can be owned by government and NGOs(acting on behalf of citizens) which can act as a neutral third parties

Electronic Health Records (EHR) application using Hyperledger Fabric

Problem Statement

- Design a decentralized system to capture medical records
- Integrity of data should be maintained
- Preserving Patients privacy: Sensitive data like prescription dosage and Bill amounts must be encrypted with patient's public key only
- Ownership of Data: Patient should decide who could access his data
- Complete Healthcare ecosystem covering Patients, Doctors/Labs, Insurance companies, pharmacies and other health-care institutions.

Organizations

- Hospital - Doctors, Labs and Nurses
- Institution - Patients
- Insurance
- Pharmacy

Assets

Appointment

+ Requested Time
+ Notes
+ Status
+ Disease Classification
+ Insurance Flag
+ Insurance Plan ID
+ Patient Public Key
+ Insurance Company ID
+ Hospital ID
+ Owner ID

Prescription

+ Status
+ Dosage
+ Reports (array)
+ Report Invoker ID's (array)
+ Requestor Public Key
+ Re-Encryption Key (array)

Bill

+ Amount
+ Status
+ User Approval
+ Owner
+ Requestor Public Key
+ Re-Encryption Key

Insurance Plan

+ Plan Type
+ Status
+ Owner
+ Insurance Company ID
+ Allocated Insurance ID
+ Insurance Public Key

Transactions
(some)

1. requestAppointment
2. acceptAppointment
3. getAppointmentStatus
4. generatePrescription
5. getPendingPaymentRequests
6. registerInsurance
7. acceptInsuranceRegistrationRequest
8. allowPrescriptionOthers
9. getBillStatus
10. getInsurancePaymentRequests

Transaction Flow

Process View

Deployment

PatientORG
+ Orderer1
+ peer0
+ CA
+ Fabric Rest API
Server1
HosptialORG
+ Root Orderer
+ Orderer0
+ peer0
+ CA
+ Fabric Rest API
+ Hyperledger Explorer
Server0
Common Channel
Raft Consensus
InsuranceORG
+ Orderer2
+ peer0
+ CA
+ Fabric Rest API
Server2
PharmacyORG
+ Orderer3
+ peer0
+ CA
+ Fabric Rest API
Server3

Conclusion

- Architecture can ensure patient's medical record privacy
- Integration of various healthcare service provider
- Helpful for efficient management of medical records
- Records Integrity maintained by Blockchain

Blockchain Technology And Applications

Sandeep K. Shukla

IIT Kanpur

C3I Center

Prevailing misconceptions about Blockchain #1

- **Blockchain is like the Internet** – you need to build a network infrastructure around the country or in a state – so that everyone can use that infrastructure to do their e-governance and other applications.

Prevailing misconceptions #2

- **Blockchain is secure**

Prevailing misconceptions #3

- **Cryptography guarantees Blockchain data integrity**

Prevailing misconceptions #4

- **Government must design a blockchain on its own and force every government organization to use its implementation of blockchain**

Prevailing misconceptions #5

- **One has to have a government vetted closed source blockchain platform on which to develop e-governance solutions**

Prevailing misconceptions #6

- **Blockchain cannot have data-privacy**

Prevailing misconceptions #7

- **“Right to forget” in the data privacy law (being tabled in the parliament) is incompatible with blockchain usage**

Prevailing misconceptions

- **Data localization is not guaranteed if we use off the shelf blockchain**

Prevailing misconceptions #9

- **Government must regulate blockchain technology**

Prevailing misconceptions #10

- Blockchain is all about crypto-currency