Blockhains and Distributed Ledgers Lecture 08

Aggelos Kiayias

THE UNIVERSITY of EDINBURGH

Lecture 08

- Permissionless vs. Permissioned Ledgers.
- BFT and PoS-based approaches for permissioned ledgers.
- Dynamic participation.

Permissionless Protocols

- Bitcoin and similar PoW-based blockchain protocols provide a permissionless setting:
  - Anyone can participate in the protocol and receive BTC as rewards by performing the PoW-based mining operation.
  - The mechanism of pouring currency in the system via proof of work, makes it feasible for anyone (possessing sufficient hashing power) to participate.
  - The ledger itself is public, readable and writeable by anyone (the latter assuming one possesses bitcoin)

Permissioned Protocols

- Participation is restricted:
  - Producing transactions and/or blocks can only be performed after being authorized by the other nodes.
- In their simplest form the set of nodes is **static**: the set of nodes implementing the protocol is fixed and determined at the onset of protocol execution.

Permissioning How-To

- Most straight approach: employ a PKI (=public-key infrastructure).
- Based on digital signatures / authentication protocols.
- **Certificate authorities** can authorize other entities.
    - authorization includes a signature from the CA on the entity's public-key, identity information etc.
- Sharing certificate authority information is necessary; (how, where?)

X.509 Certificates

- Internet standard since 1988.
- Hierarchical.
  - http://www.ietf.org/rfc/rfc3280.txt

Structure of X.509 Certificates

Version
Serial Number
Algorithm / Parameters
Issuer
Period of Validity: not before date not after date
Subject
Algorithm/ Parameters/ Key
x509v3 extensions
...
Signature

X.509
does not
specify
cryptographic
algorithms

Example, I

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            13:86:35:4d:1d:3f:06:f2:c1:f9:65:05:d5:90:1c:62
        Signature Algorithm: sha1WithRSAEncryption
        Issuer: C=US, O=VISA, OU=Visa International Service Association, CN=Visa eCommerce Root
        Validity
            Not Before: Jun 26 02:18:36 2002 GMT
            Not After : Jun 24 00:16:12 2022 GMT
        Subject: C=US, O=VISA, OU=Visa International Service Association, CN=Visa eCommerce Root
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (2048 bit)
                Modulus (2048 bit):
                    00:af:57:de:56:1e:6e:a1:da:60:b1:94:27:cb:17:
                    db:07:3f:80:85:4f:c8:9c:b6:d0:f4:6f:4f:cf:99:
                    d8:e1:db:c2:48:5c:3a:ac:39:33:c7:1f:6a:8b:26:
                    3d:2b:35:f5:48:b1:91:c1:02:4e:04:96:91:7b:b0:
                    33:f0:b1:14:4e:11:6f:b5:40:af:1b:45:a5:4a:ef:
                    7e:b6:ac:f2:a0:1f:58:3f:12:46:60:3c:8d:a1:e0:
                    7d:cf:57:3e:33:1e:fb:47:f1:aa:15:97:07:55:66:
                    a5:b5:2d:2e:d8:80:59:b2:a7:0d:b7:46:ec:21:63:
```

Hierarchical name spec

The RSA Algorithm is used; the DSA is also an option

This is a self-signed root certificate

Example, II

```
        4a:ea:db:df:72:38:8c:f3:96:bd:f1:17:bc:d2:ba:
        3b:45:5a:c6:a7:f6:c6:17:8b:01:9d:fc:19:a8:2a:
        83:16:b8:3a:48:fe:4e:3e:a0:ab:06:19:e9:53:f3:
        80:13:07:ed:2d:bf:3f:0a:3c:55:20:39:2c:2c:00:
        69:74:95:4a:bc:20:b2:a9:79:e5:18:89:91:a8:dc:
        1c:4d:ef:bb:7e:37:0b:5d:fe:39:a5:88:52:8c:00:
        6c:ec:18:7c:41:bd:f6:8b:75:77:ba:60:9d:84:e7:
        fe:2d
      Exponent: 65537 (0x10001)
  X509v3 extensions:
    X509v3 Basic Constraints: critical
      CA:TRUE
    X509v3 Key Usage: critical
      Certificate Sign, CRL Sign
    X509v3 Subject Key Identifier:
      15:38:83:0F:3F:2C:3F:70:33:1E:CD:46:FE:07:8C:20:E0:D7:C3:B7
Signature Algorithm: sha1WithRSAEncryption
  5f:f1:41:7d:7c:5c:08:b9:2b:e0:d5:92:47:fa:67:5c:a5:13:
  c3:03:21:9b:2b:4c:89:46:cf:59:4d:c9:fe:a5:40:b6:63:cd:
  ..............
```

note:
public-exponent, corresponding to RSA function: the public-key is a large integer N that is a product of two primes and a small exponent.

Note that SHA-1 is broken now : https://shattered.io

Certificate Encoding

- Certificates are binary files.
- They can be Base64 encoded to be stored/tansported as text files.
- This encoding is called PEM from (privacy enhanced mail)


```
-----BEGIN CERTIFICATE-----
MIIFXzCCBEegAwIBAgIHK5Bmkq9qPzANBgkqhkiG9w0BAQUFADCByjELMAkGA1UE
BhMCVVMxEDAOBgNVBAgTB0FyaXpvbmExEzARBgNVBAcTClNjb3R0c2RhbGUxGjAY
BgNVBAoTEUdvRGFkZHkuY29tLCBJbmMuMTMwMQYDVQQLEypodHRwOi8vY2VydGlm
aWNhdGVzLmdvZGFkZHkuY29tL3JlcG9zaXRvcnkxMDAuBgNVBAMTJ0dvIERhZGR5
IFNlY3VyZSBDZXJ0aWZpY2F0aW9uIEF1dGhvcml0eTERMA8GA1UEBRMIMDc5Njky
ODcwHhcNMTEwMTEwMTkyMzIyWhcNMTIwMjI1MTMyMTEyWjBZMRkwFwYDVQQKDBAq
LmVuZ3IudWNvbm4uZWR1MSEwHwYDVQQLDBhEb21haW4gQ29udHJvbCBWYWxpZGF0
ZWQxGTAXBgNVBAMMECouZW5nci51Y29ubi5lZHUwggEiMA0GCSqGSIb3DQEBAQUA
A4IBDwAwggEKAoIBAQCOgAbFE4TOs2KA/WM9rZ5Q+3G6teSFdjmxZynMeCch+/e/
bSudinQ0oJShqXkyiwrGa1brr3pbQkGLn7AUdli+479IMOq2tFpvZ9SgCVsp2Jk1
/6MqBleBTKfVGfVEz6+YzkwcoiEe7cY+deOeIG1ZFVlnI3kw19V42gk+ZU25oVgL
uiJB2RRD3djpaV1txMVyCnzHa0u2eEHhmh87faIA6EckEW/cWRoVuBZF5Oojsr5c
iXgLIwImQn+Cmo7NQVGV4pKqJVnHPyFX8w+CcLJ8GonVxQeKlehO2DycpDoi2SpP
/hAZ1Vh9mEfftpCikXPwF1unoZAmaN9mWUtcpyMnAgMBAAGjggG4MIIBtDAPBgNV
HRMBAf8EBTADAQEAMB0GA1UdJQQWMBQGCCsGAQUFBwMBBggrBgEFBQcDAjAOBgNV
HQ8BAf8EBAMCBaAwMwYDVR0fBCwwKjAooCagJIYiaHR0cDovL2NybC5nb2RhZGR5
LmNvbS9nZHMxLTI4LmNybDBNBgNVHSAERjBEMEIGC2CGSAGG/W0BBxcBMDMwMQYI
KwYBBQUHAgEWJWh0dHBzOi8vY2VydHMuZ29kYWRkeS5jb20vcmVwb3NpdG9yeS8w
gYAGCCsGAQUFBwEBBHQwcjAkBggrBgEFBQcwAYYYaHR0cDovL29jc3AuZ29kYWRk
eS5jb20vMEoGCCsGAQUFBzAChj5odHRwOi8vY2VydGlmaWNhdGVzLmdvZGFkZHku
Y29tL3JlcG9zaXRvcnkvZ2RfaW50ZXJtZWRpYXRlLmNydDAfBgNVHSMEGDAWgBT9
rGEyk2xF1uLuhV+auud2mWjM5zArBgNVHREEJDAighAqLmVuZ3IudWNvbm4uZWR1
gg5lbmdyLnVjb25uLmVkdTAdBgNVHQ4EFgQU9Z+3hgNCeZLUiGGdUr4pyrMtS4cw
DQYJKoZIhvcNAQEFBQADggEBAGJzGVObEDODThT3di8+JC8hKde1DxOHNzmtv3I3
/eDGR5vO8squzmWh9fn5FZFbHd/UfFPddO6ykR7ffPLGTNCUYbviVjfA+hTpPnPA
3D5u791VfPUJBZf+ccAGr9E1BY+kYGaGqJgHmHcdG7XU0d7bU0XW82Q8wbHQj7H4
jvKSrir4bdGbmZ5VMAjsRL7xMwVypbh2KlOq/wU6AUDs3Rodawa6BnqyCHEpjn0I
i2NUiTzDRKL2OvRCPNs7aY1s4L0oN+HU6yeDkriOHR1NBt3lBGLW+bjZalT6GjPl
xz4fZtag71g7a9mgSAwDE9YlCP2e9WSdlf0THZQ5y3aV868=
-----END CERTIFICATE-----
```

Binary Encoding

- The elements of certificate are encoded using DER (distinguished encoding rules)

Items are stored in TLV format : triplets <Type,Length,Value>

```
1.   30 23          ; SEQUENCE (23 Bytes)
2.   | | 31 0f          ; SET (f Bytes)
3.   | | | 30 0d          ; SEQUENCE (d Bytes)
4.   | | |    06 03         ; OBJECT_ID (3 Bytes)
5.   | | |    | 55 04 03
6.   | | |    |    ; 2.5.4.3 Common Name (CN)
7.   | | |    13 06         ; PRINTABLE_STRING (6 Bytes)
8.   | | |       54 65 73 74 43 4e                ; TestCN
9.   | | |          ; "TestCN"
10.  | | 31 10          ; SET (10 Bytes)
11.  | |    30 0e          ; SEQUENCE (e Bytes)
12.  | |       06 03         ; OBJECT_ID (3 Bytes)
13.  | |       | 55 04 0a
14.  | |       |    ; 2.5.4.10 Organization (O)
15.  | |       13 07         ; PRINTABLE_STRING (7 Bytes)
16.  | |          54 65 73 74 4f 72 67              ; TestOrg
17.  | |             ; "TestOrg"
```

Digital Signatures and Certificates

- A certificate contains a digital signature.
- Recall that cryptographic design of digital signatures involves typically:
  - A cryptographic signing operation that acts on a fixed input of a specific type and has a public-verifiability feature.
  - A cryptographic hash function that takes arbitrary strings and maps them to the data type suitable for the signing operation.
- Common setting today: SHA2 with RSA/DSA.

Certificate Considerations

- All computer systems come with preloaded certificates from certificate authorities. This provide a **setup assumption**.
- Certificates need to be revoked in case the corresponding secret keys become exposed or the algorithms used are not safe anymore.
- In a blockchain system, certificate information can be provided as part of the genesis block.

Secure Channels and Certificates

- Possession of mutually acceptable certificates not only permits authenticated communication (exchanging signed mechanism between two entities) but also allows building a secure channel

  - Protocol **TLS 1.2** is used to build such secure channel.

  - It relies on cryptographic protocols such as the Diffie Hellman key exchange. It can ensure the confidentiality of the data exchanged.

Static Permissioned Blockchain

- All participants are identified by self-signed certificates in the genesis block.
- The set of participants remains the same throughout the execution.
- This is the simplest form of a PKI / public-key directory.

Recall: Robust Transaction Ledger

- persistence: Transactions are organized in a “log” and honest nodes agree on it.
- liveness: New transactions are included in the log nodes, after a suitable period of time.

In more detail

- Common prefix / persistence / consistency of the LOG:
  - Each node maintains a log, and at any two times t1 <= t2, it holds that LOG1 is a prefix of LOG2.
- Liveness of the log:
  - An honestly generated transaction will become part of all honest parties' LOGs after a specified time window *u.*

Permissioning

- Prior to system operation the nodes register their certificates that are included in the genesis block.
- Using such certificates, all the nodes are capable of authenticating each participant and allowing interaction with the LOG in a way that is prescribed by the participants' credentials.

Centralized Permissioned Ledger

- One of participants acts as a server and maintains the LOG.
- Readers and writers to the LOG authenticate with the server and can perform read and write operations.
    - Consistency of the LOG is guaranteed assuming the server is trusted.
    - Liveness of the LOG is guaranteed assuming the server is trusted and functional.

Bitcoin's Permissionless Setting

- The genesis block contains no certificate information.
- Reading from the LOG is open (anyone can do it without credentials).
- Writing to the LOG can only be done in specific ways (issuing transactions).
  - Nodes can obtain valid credentials (accounts) by generating a public and secret-key and either mining a block (which will reward their account with BTC) or buy BTC from another node.
  - Once the LOG records their account credit, they can issue transactions.
  - In essence: crediting a bitcoin account is creating a certificate that imparts the account holder with certain permissions w.r.t. the ledger.

Distributed Permissioned Ledger, I

- A number of servers maintain the ledger LOG individually.

- Each share the same genesis block that identifies all participants.

- Assuming a synchronous operation, at each round, Readers and Writers authenticate with the servers and interact with the LOG in a prescribed fashion.

Distributed Permissioned Ledger, II

- Readers authenticate to each server and obtain Read access.
- Writers authenticate to each server and provide their inputs.
- Servers run a **consensus protocol** to agree what inputs should be included in the LOG.

Reader/Writer Management

- Readers and Writers can authenticate to each server referring to the information in the genesis block.
- It is possible to introduce additional readers and writers by suitably issuing certificates to other users.
- Note that each participant would then need to show a valid certificate chain that establishes her privileges for the specific read or write access that is requested.

Read requests

- Is it possible to restrict read requests as in the centralized setting?
  - Nodes can maintain blocks of transactions private and issue them only to users that are authenticated.
  - The TLS protocol can be used to build a secure channel between the reader and the responding node.
- Note that the above would require that all servers remain honest (as they all share the LOG).

BFT-based approach for permissioned ledgers

- Focus on write requests next. We want to ensure LOG liveness and consistency.
- We will apply a “byzantine fault tolerant” (BFT) agreement protocol that uses two important tools:
  - a graded broadcast.
  - and a common coin binary consensus protocol.

Graded Broadcast

- Parties involved : a single sender and several receivers.
  - The i-th receiver outputs (M_i, G_i).
  - The value G_i is in {0,1,2}.
    - If the sender is honest then $M_i=M_j$ for all i,j and $G_i=2$.
    - If the sender is malicious and one receiver outputs $G_i=2$ then other honest receivers output $G_i>=1$.

Graded Broadcast Construction, I

- **Round 1.** The sender sends the message *M* to all receivers.

- **Round 2.** The *i*-th receiver obtains *M1i* from round 1 and sends it to all receivers.

- **Round 3.** The *i*-th receiver obtains *M2ji* from the *j*-th receiver in round 2 and performs the following:
  if there is a single message that was sent by *2n/3* receivers then send it to all receivers. Else send nothing.

Graded Broadcast Construction, II

- The *i*-th receiver obtains *M3ji* from the *j*-th receiver in round 3.
    - If there is a single message that was sent by more than *2n/3* receivers output that message as *Mi* and set *Gi*=2.
    - If there is a single message that was sent by more than n/3 receivers output that message as *Mi* and set *Gi*=1.
    - In any other case output *fail* and *Gi*=0.

Graded Broadcast Construction, III

- Analysis. Assume that malicious parties are $t < n/3$.
  - If the sender is honest, then each receiver will receive the same message $>= 2n/3$ times in round 2 and 3. All honest receivers will output *Gi=2* and that message.

Graded Broadcast IV

- **Lemma.** If two honest receivers send a message in round 3 it **must be** the *same.*

  - Indeed, if they send messages $M \neq M'$, they both have received them by at least 2n/3 receivers from round 2.

  - Given the above, observe that $2n/3-t>n/3$ honest parties have sent *M* in round 2.

  - Thus <n-n/3 = 2n/3 parties are capable of sending M' (which is different than *M*), leading to contradiction.

Graded Broadcast, V

- Suppose the *i*-th receiver returns $G_i=2$ and let M_i be the message it chooses. Consider the output (M_j, G_j)
  - The *i*-th receiver has received the message from at least $2n/3$ receivers in round 3.
  - => More than $n/3$ honest receivers have sent M in round 3. Thus the *j*-th receiver should produce a grade $G_j>=1$.
  - If there is another message M' sent by more n/3 receivers in round 3, at least one of them is honest; this leads to a contradiction of the **lemma**. Thus $M_j=M_i$.

From Graded Broadcast to Consensus

- Execute n/3+1 phases:
  - In each phase perform:
    - A graded broadcast by a designated sender.
    - A consensus protocol to determine whether everyone's grade = 2. If that is true terminate with the output of the graded broadcast. Each node signs the output to generate a public endorsement.

Recall Consensus Properties

- Termination $\forall i \in \mathsf{H}(u_i \text{ is defined})$
- Agreement $\forall i, j \in \mathsf{H}\,(u_i = u_j)$
- Validity $\exists v(\forall i \in \mathsf{H}\,(v_i = v)) \implies (\forall i \in \mathsf{H}\,(u_i = v))$

Common Coin Consensus, I

- **Round 1.** The *i*-th party sends *Bi* to all parties.
- **Round 2.** A randomly selected common coin *c* is made available.
- **Round 3.** The *i*-th party obtains *B1i* from round 0 and performs the following:
  if 0 was sent by *2n/3* parties set *B=0.*
  if 1 was sent by *2n/3* parties set *B=1.*
  else set *B=c.*

Common Coin Consensus, II

- **Validity.** Suppose all honest parties agree on their input initially; i.e., for all *i*,*j*, $B_i = B_j$.
  - Then, given that $t<n/3$, they will set *B* to that bit in round 3.
- **Agreement.** Suppose that an honest party outputs 0 because *2n/3* sent 0 in round 1. It follows that $> n/3$ honest parties had 0 as input prior to the protocol. As a result any other honest party will also output 0 with probability 1/2. Similar argument can be made for 1.
- **Termination.** it will always happen in 3 rounds.

Common Coin Consensus, III

- Repeat protocol for *k* times.
- After *k* repetitions, it is guaranteed that honest parties will reach agreement with probability 1-2^(-k).
  - (note that they will not know they did).
  - [more involved techniques exist that bring the number of repetitions to expected constant with detection of agreement].

Producing a common coin

- Suppose that the common coin protocol is imperfect and serves a proper coin with probability ε (no other assumptions otherwise)
  - (e.g., with probability 1-ε, the parties may produce a different adversarially selected value).
- Then, agreement in the consensus protocol will only hold with probability ε/2.
- Same analysis as before works but need to repeat protocol for kε^{-1} times.
  - It is easy to implement common coin for small ε: have each party simply flip a coin. it holds that ε = 2^{-n+t+1}. [More involved techniques provide a large constant ε<1]

Distributed Ledger

- Using the above protocol it is guaranteed that:
  - The LOG will grow by a new set of transactions.
  - The transactions are contributed by an honest server and thus include all new transactions received by that node.
  - The waiting time *u*, for a transaction to be included is proportional to the termination of the protocol that agrees on the next LOG entry; (exponential in our example but can be improved to constant).

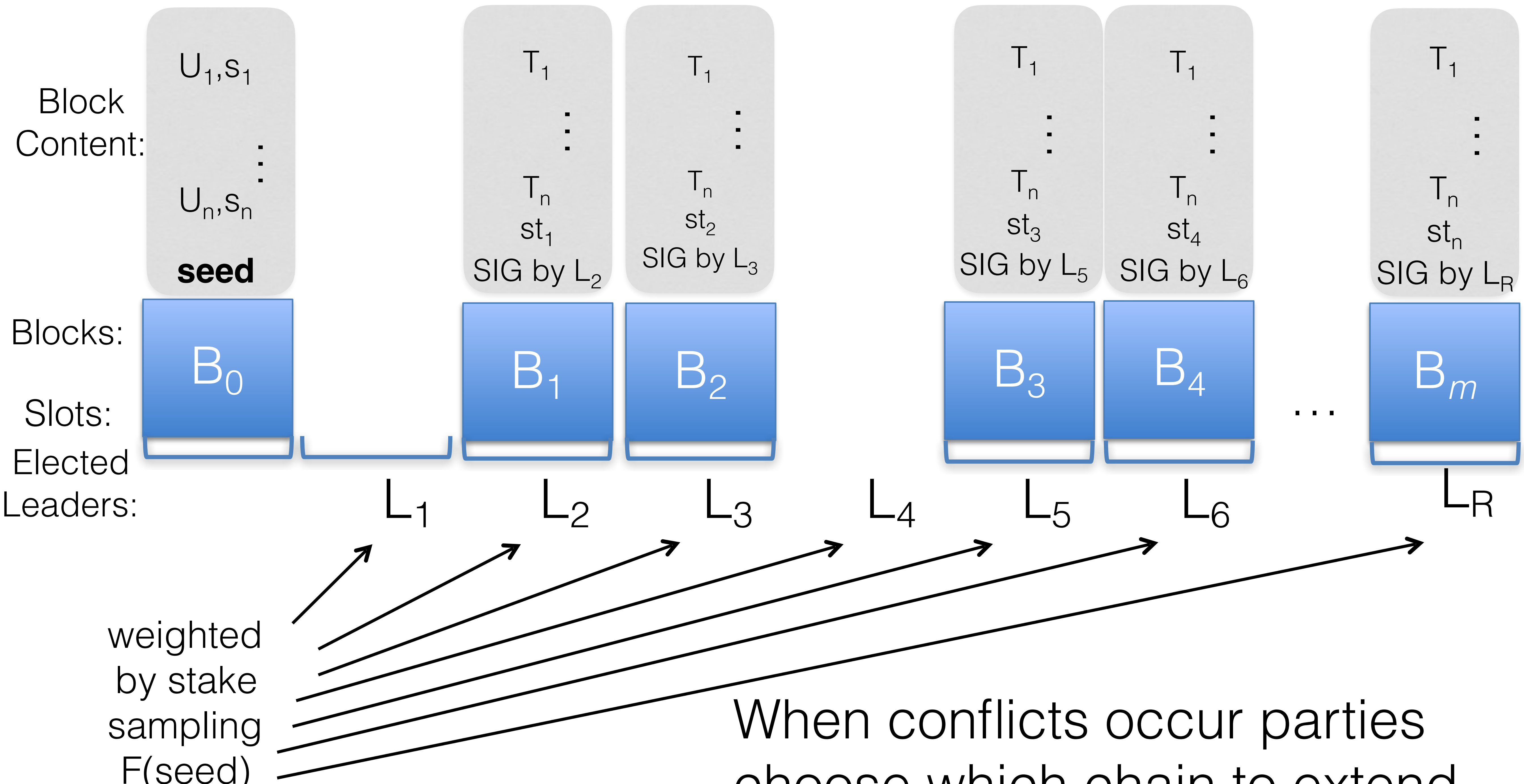
PoS-based Approach for Permissioned Ledgers

- Recall Proof of stake (PoS) protocols generalize the bitcoin blockchain protocol. Substitute PoW for PoS.

- Resulting protocol resembles blockchain operation (as opposed to the BFT approach).

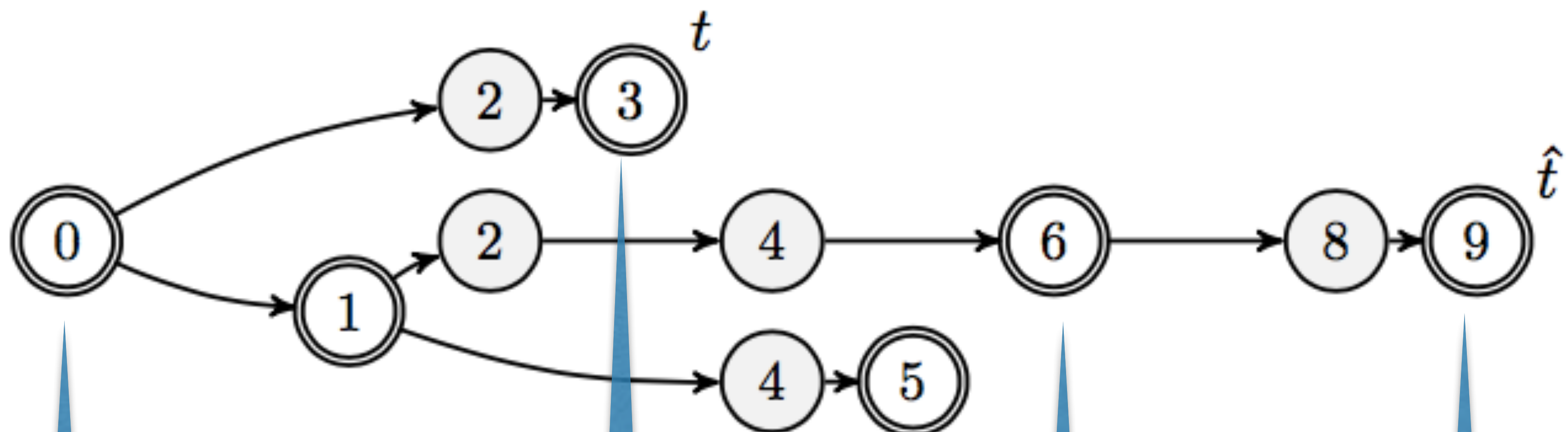
    - Example: the Ouroboros protocol.

Ouroboros : Static Stake



When conflicts occur parties choose which chain to extend based on **longest chain rule**

Forks and Protocol Executions



Characteristic string $w = 0\ 1\ 0\ 1\ 0\ 0\ 1\ 1\ 0$

$$w_i = \begin{cases} 0 & i\text{-th slot belongs to an honest party} \\ 1 & i\text{-th slot belongs to a malicious coalition} \end{cases}$$

Genesis

honest party produce a block 1

adversary serves block 2 to honest party

adversary serves block 4 to honest party

adversary serves block 2,4 to honest party

adversary serves block 8 to honest party

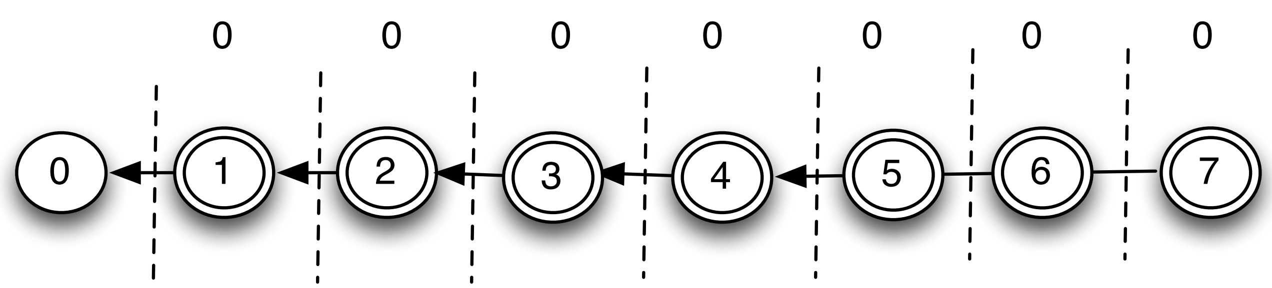
PoW vs. PoS Proof

- The adversary is at a **much better position** in this protocol execution compared to Bitcoin's PoW-based execution.
  - it can **see** ahead of time how stakeholders are activated.
  - it can **generate multiple** different blocks for the **same slot** at any time **without cost**.
  - it can **wait** and act just before an honest party comes online.

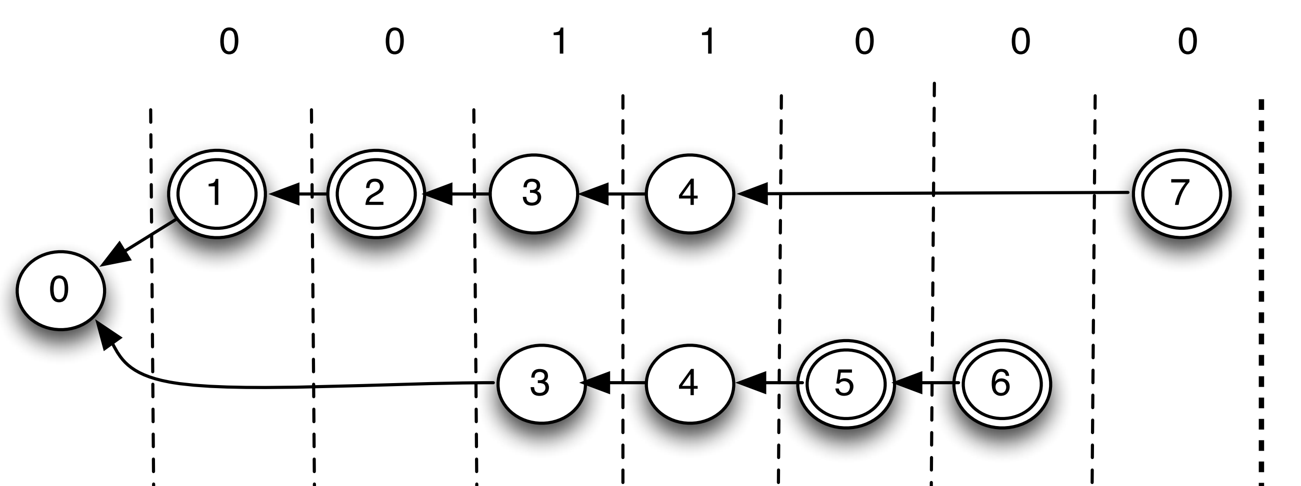
Forkable Strings

The characteristic strings the adversary prefers!

Not Forkable!



Forkable!



the $t<n/3$ setting

- Consider any string with Hamming weight less than $n/3$.

- **Lemma.** A slot controlled by the attacker can be used in at most two different locations to contribute blocks in the set of all admissible blockchains (those that correspond to protocol executions).

- If follows if $t<n/3$, there is not enough slots possessed by the adversary to match the blocks of the honest participants.

Permissioned Ledgers with Dynamic participation

- Participation is still restricted: Producing transactions and/or blocks can only be performed only after being authorized.

- But dynamic: set of nodes implementing the protocol is evolving over time.

- Each participant's relative weight can be expressed by the **stake** they have in the system. Participants = Stakeholders.

- Buying stake in the system may be performed via an exchange, thus the entry bar to the system is low (but still not as open as the permissionless setting of bitcoin).

Design Approach

- Iterate a basic recursive step:
  - Form a committee that will run a fixed number (1 or more) of permissioned protocol executions.
  - The relative weights of the committee members should be selected according to relative stake of all stakeholders.
  - A condition on honest stake should translate to a suitable committee restriction.
- Update the stake of each party based on the outcome of the inputs and repeat ad infinitum.

Forming a committee, I

- The elected committee to perform the permissioned protocol execution should respect the adversarial bound under which the underlying protocol execution operates.

- Example of the simplest possible case: the stakeholder distribution is flat; a bound of $t<n/3$ is required. The committee selected is the set of all stakeholders.

- A bound of $<1/3$ in terms of relative adversarial stake directly translates to a $t<n/3$ bound in the selected committee.

Forming a committee, II

- Consider a biased stakeholder distribution.
- Where the i-th stakeholder has stake s_i with total stake is $S \in \mathbb{N}$
- Execute the permissioned protocol with the i-th stakeholder maintaining s_i identities out of a total space of identities that has cardinality equal to S.

Forming a committee, III

- What if the size of the total stake S is prohibitively large?
  - Random sampling may be used to select a subset of stakeholders to form the committee.
  - Perform the sampling following “weighing by stake”: the probability that the i-th stakeholder is selected should be s_i / S.
  - Randomness should be refreshed in each permissioned execution.

Stake Shift

- The stake shift signifies the statistical distance between the random weighed by stake sampling of a stakeholder between two stakeholder distributions.

- E.g., if a stakeholder distribution has 3 members sharing equal stake and one of them is replaced, the stake shift is 1/3.

- The maximum stake shift is 1 (the set stakeholders have completely changed).

Stakeholder majority

- As the permissioned execution advances the stake shifts, since transactions are posted.

- The assumption that there is an **honest stakeholder majority** "at present" may not imply necessarily that this can imply the security of the execution.

- **Bounding** stake shift over the permissioned execution enables to use a current honest stakeholder majority to infer a honest stake holder majority in the past.

Refreshing Randomness

- Critical for security :
  - Possible via techniques from secure multiparty computation. Coin flipping protocols.
  - Or derivation from some aspect of the protocol execution (e.g., hash the previous block of transactions and use the output as coins to seed the sampling).
- Beware: **grinding attacks;** the attacker tries to impose a protocol history that favors adversarial parties.

New Parties Joining

- When a new party joins the system what information should be made available?
    - necessary: the initial stakeholder distribution.
    - sufficient: the most recent block for which consensus has been reached (checkpointing).

Genesis Block Joining

- The setting when new parties joining have access only to the genesis block.
- One has to deal with a "**long range**" attack:
    - Attacker builds a LOG that is acceptable starting from the genesis block.
- New parties joining have the task to choose which is the correct version of the LOG.

An example of a long range attack

- Assume the adversary can break with probability ε the randomness generation and choose the outcome.

  - Then it is impossible to protect against a long range attack: over an expected period of ε^(-1) permissioned executions, the attacker will get the opportunity to transfer control to a "**sybil**" committee.

  - (A sybil attack is when an attacker creates multiple fake identities that make the attacker appear as multiple parties in the system).

Mitigating Long Range Attacks

- ensure good randomness generation (if random sampling is used).
- Identify characteristics in the LOG structure that signify that the LOG is not the correct one. For instance:
  - blocks are signed by too few of the stakeholders in the BFT-based approach.
  - the sequence of blocks is too sparse in the time-domain in the PoS-based approach.

End of lecture 08

- Next lecture:
  - Pitfalls and security vulnerabilities in smart contracts.