A Distributed X.509 Public Key Infrastructure Backed by a Blockchain

A public key infrastructure (PKI) is a set of roles, policies, and procedures needed to create, manage, distribute, use, store, and revoke digital certificates and manage public-key encryption.

Keywords: Blockchain, Digital signatures, Merkle tree, PKI, Proof of Stake, X.509 Certificate

Understanding the problem

- A **certificate** (in the X.509 format) binds a name to a public key which can be used for authentication.
- On the web, a certificate binds a DNS domain name to a public ECDSA/RSA key.
- Certificates are digitally signed by trusted certificate authorities.
- *Certificate authorities sometimes fail in their duty and issue a fraudulent certificate, which can be used to perform a man-in-the-middle attack.*

It is hard to run a certificate authority!

- In 2011:
  - The DigiNotar CA files for bankruptcy after being compromised by an attacker who issued over 500 certificates for different domains.
  - Comodo RAs tricked into signing bogus certificates for Skype, Yahoo, Mozilla and Google's Gmail.
  - TürkTrust signs two intermediary certificates by accident. One of them is revoked immediately, the other one gets revoked a year later…
- In 2013:
  - The French Treasury DG Trésor signs a fake Google certificate to read workers' traffic.
- In 2015:
  - Symantec's subsidiary Thawte generates over 187 fake certificates for "testing purposes". Thawte fires technicians and installs an "additional tool, policy and process safeguard" to avoid further mishap.
- In 2016:
  - Mozilla will no longer trust certificates issued by WoSign and its newly acquired subsidiary StartCom, after after evidence comes to light that they have backdated CA certificates to avoid SHA-1 browser warnings.
- In 2017:
  - Google says they will no longer recognise EV certificates by Symantec, after another 108 fake certificates are detected.

How is this problem addressed today?

- **Certificate Transparency Logs** [RFC6962] Public logs containing all issued certificates. If a fraudulent certificate is issued, it will (probably) be detected.
- **DNS Certificate Authority Authorisation** [RFC6844] A signed DNS record containing the certificate authorities who are allowed to issue certificates for a domain. Must be honoured by all certificate authorities from September 2017.
- **DNS Based Authentication of Named Entities** [RFC6698] Signed DNS records which contains information about how a TLS client should verify a certificate.
- **Public Key Pinning** [RFC7469] allows a server to send a hash of one of the public keys in the certificate chain to the client. This public key is then cached by the client, and subsequent connection attempts will fail if the public key has changed.

Can we do better?

Idea: Build a decentralised PKI. Use certificate authorities to approve new accounts, use a blockchain for consensus.

- Every domain holder administers its own keypair (K_{pub}, K_{priv}) in this decentralised PKI.
- Certificates issued by a certificate authority must be cross-signed by K_{priv}.
- A client (for example a web browser) lookups K_{pub} on the blockchain to validate the certificate received from the server.

Research question

Are blockchains suitable as a building block for a distributed public key infrastructure and what could such a public key infrastructure look like?

The account tree (1/2)

- Stores a mapping `{Name} ↦ {Public Key}`.
- An authenticated data structure with membership proofs - allows us to convince a web browser that a certain (name, public key)-pair exists in the tree!

Each node contains an account:

```
struct account {
    domain name,
    public key,
    expiry date,
    locking script,
}
```

The account tree (2/2)

The account tree T implements the following methods:

`Lookup(Name) ↦ (Account, Proof)`: Returns an account A and a membership proof for A iff A ∈ T.

`Insert(Account)`: Inserts the account A into T.

`Delete(Name)`: Deletes the account A from T.

`Prune(Date)`: Deletes all accounts with an expiration date $D_A \leq$ Date.

Consensus problem

Which version of the account tree is the “correct” one?

We need to solve this problem to be able to verify certificates in a secure manner.

Reasons why blockchains are interesting:

1. **No central authority** which needs to be trusted.
2. **Transparency** - cheating will be detected.
3. **Bitcoin** [Nak08] - solves the same problem but for money. Have successfully been in operation for about 8 years.

Transactions and blocks

- A ***transaction*** (Insert, Update, Delete) is a transformation

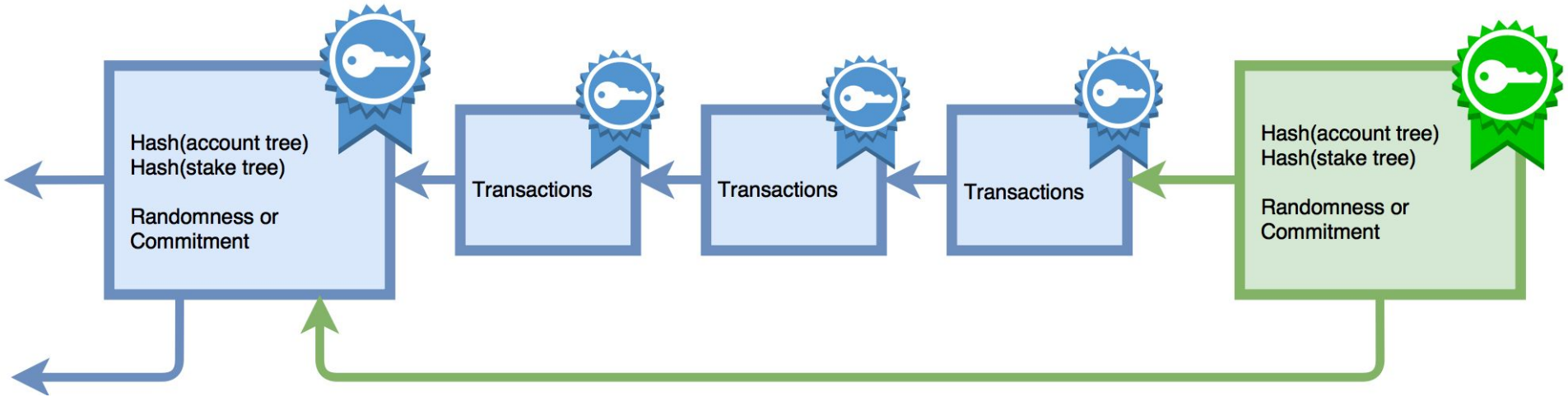
  `{Account tree, Auxiliary data} ↦ {Account tree}`
- Every transaction must be approved (signed) by a certificate authority.
- Transactions are packed into ***blocks*** which can be chronologically ordered in a chain.
- Two types of blocks: **microblocks** and **keyblocks** as in [Eyal16].
- The “correct” account tree is the account tree backed by the blockchain with the most keyblocks.

Stakeholders

- Blocks are added to the blockchain by semi-trusted entities called “stakeholders” stored in a *stake tree*.
- A stakeholder in the stake tree controls an amount of coins and a private key used to sign blocks.
- Stakeholders can be added or removed from the stake tree through voting.

Time slots

A stakeholder is allowed to add blocks to the blockchain during its designated *time slot*.

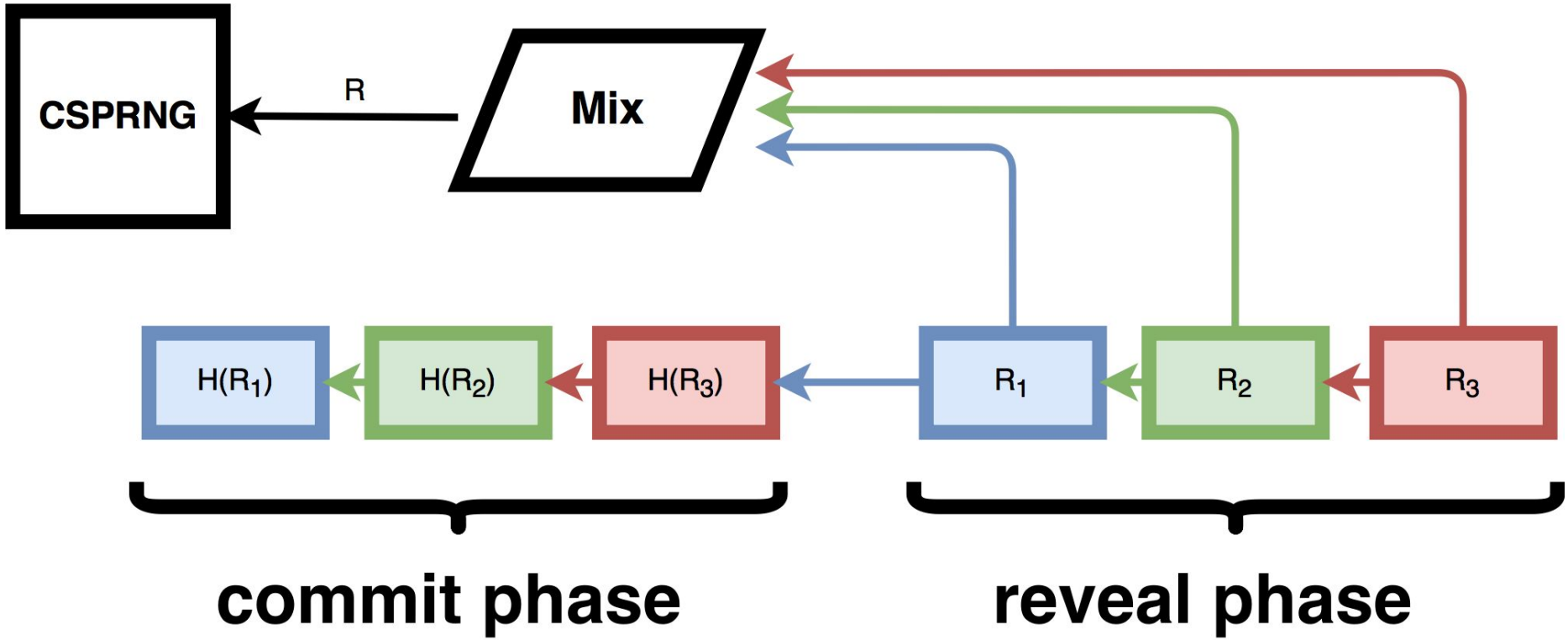


Electing block leaders (1/3)

How are time slots allocated?

1. Time is chopped up into epochs.
2. Each epoch is divided into a commit phase and a reveal phase.
3. A stakeholder is given two time slots, one during the commit phase, and one during the reveal phase.
4. Stakeholders for the an epoch are chosen jointly by the stakeholders from the previous epoch.

Electing block leaders (2/3)



Electing block leaders (3/3)

Using the CSPRNG we can sample a set of stakeholders from the stake tree, using an idea called follow-the-satoshi.

Follow-the-satoshi: The stakeholder to sign the next block is the owner of a coin, chosen at random [Ben14].

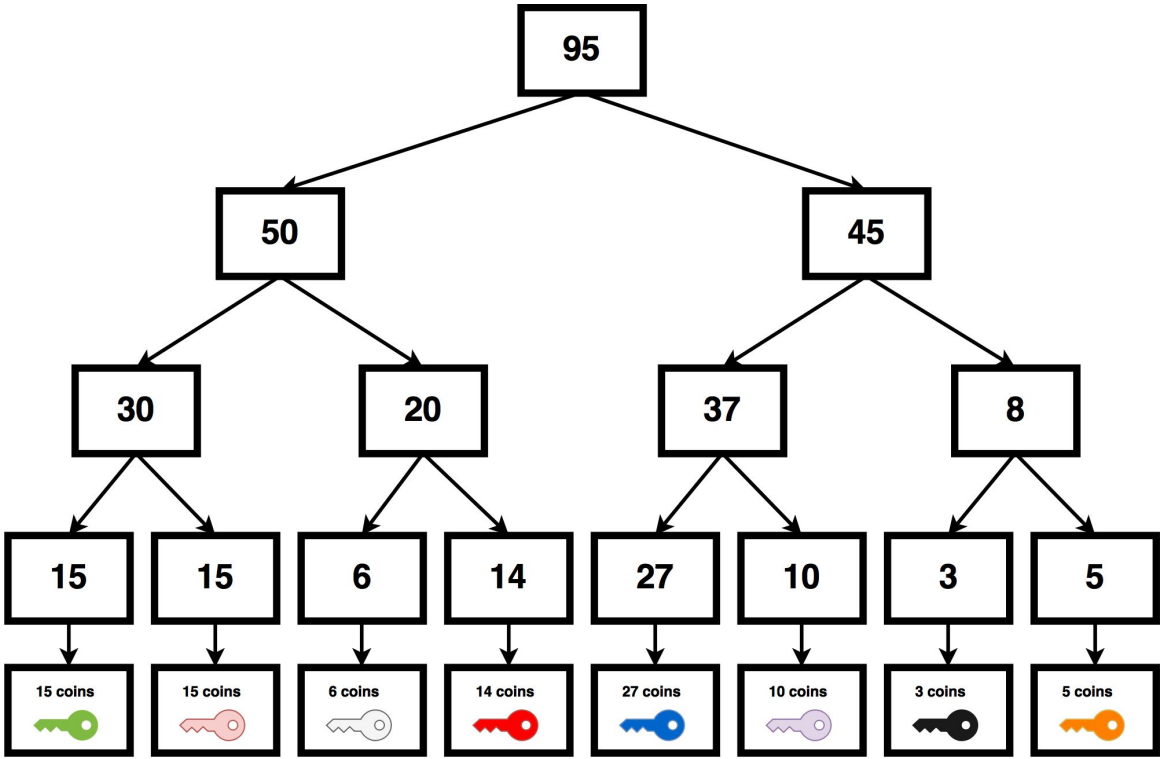
Note: Fairness required

$$P(\text{stakeholder S chosen}) = \frac{\text{coins(S)}}{\text{coins(stake tree)}}$$

Follow-the-satoshi in the stake tree

FTS-TREE (Appendix A)

1. Start at the root
2. Generate a random number r in the interval [1, coins(left)+coins(right)]
3. If 1 ≤ r ≤ coins(left) choose the left branch, otherwise choose the right branch.
4. If the subtree is leaf, return the corresponding stakeholder, else goto 2.



Voting

Can be used to:

- Adjust the maximum number of transactions per timeslot
- Add or exclude a stakeholder (punish cheating!)
- Add or revoke a certificate authorities allowed to sign transactions

Stakeholders sign their votes with their private key. A vote is linked to a keyblock on the blockchain.

Votes are only valid for a specific epoch and need to be confirmed by a coalition of stakeholders controlling a majority of the coins in the stake tree (majority decision).

Security (1/2)

Changes to the stake tree and approval of certificate authorities requires a majority decision. No security if >50% of stakeholders are malicious!

Probability for a p-coalition of malicious stakeholders to extend the blockchain with more keyblocks than the (1-p)-coalition of honest stakeholders:

$$\Pr\left(\mathbb{X} > \frac{n}{2}\right) = 1 - \sum_{i=0}^{\frac{n}{2}} \binom{n}{i} \cdot p^i \cdot (1-p)^{n-i}$$

Assumes the seed to the CSPRNG is chosen randomly!

Security (2/2)

Malicious stakeholders can ignore keyblocks by honest stakeholders to gain control over the input to the mixing function!

Stake grinding

Choose R_1, R_2… $R_{n/2}$ such that Mix(R_1, R_2… $R_{n/2}$) gives an advantage during the next epoch.

- Automatically suspend a stakeholder who creates an invalid keyblock.
- Make the mixing function slow and hard to precompute!
- Long epoch.
- Co-sign keyblocks.

Verify a certificate (Appendix G)

1. Download and verify the block headers of keyblocks belonging to the longest blockchain[1].
2. Download the public key K_{pub} corresponding to the CN in the certificate together with a membership proof P (can be supplied via OCSP[2]).
3. Validate K_{pub} using P and a hash of the account tree stored in a (recent) keyblock on the blockchain.

Note 1: This may not be possible if the connection is firewalled, e.g. on an aircraft.
Note 2: The entity providing K_{pub} and P does not need to be trusted.

Storage and bandwidth requirements

Amount of data to be processed during a time slot is estimated to be *at least* 0.52 MB/minute (corresponds to a Bitcoin block size of 5.2 MB).

What	Who	Size	Grows
Account tree	Validators and stakeholders	≈ 36 GB	≈ 2.5 GB/year
Account contents	Validators and stakeholders	≈ 185 GB	≈ 13 GB/year
Transaction history	Validators and stakeholders	≈ 22 GB/month (pruning!)	≈ 1.65 GB/year
Block headers	Everyone	≤ 24 MB/year	
Membership proofs	Web browsers	≤ 3.7 KB/certificate	

Future work and discussion

- **Switch to multiple block leaders** - Only one stakeholder at a time is allowed to confirm transactions: makes the system vulnerable to denial of service?
- **Increase throughput -** The number of transactions which can be processed every second is a bottleneck. Possible solutions:
  - Sharding
  - Limit to EV certificates
- **Improve revocation -** Revoke certificates by changing the public key on the blockchain instead of using certificate revocation lists or OCSP.
- **Ensure fair play -** Find methods to automatically detect and suspend cheating stakeholders.
- **Enable open participation** - Use Bitcoin’s Proof of Work instead of a stake tree.

References

[Nak08] NAKAMOTO, S. Bitcoin: A Peer-to-Peer Electronic Cash System, 2008.
[Ben14] BENTOV, I., LEE, C., MIZRAHI, A., AND ROSENFELD, M. Proof of Activity: Extending Bitcoin's Proof of Work via Proof of Stake. SIGMETRICS Perform. Eval. Rev. 42, 3, 2014, 34–37
[RFC6962] Laurie, Ben, Adam Langley, and Emilia Kasper. *Certificate transparency*. No. RFC 6962. 2013
[RFC6844] Hallam-Baker, Phillip, and Rob Stradling. "DNS certification authority authorization (CAA) resource record.", 2013
[RFC6698] Schlyter, Jakob, and Paul Hoffman. "The DNS-based authentication of named entities (DANE) transport layer security (TLS) protocol: TLSA.", 2012
[RFC7469] Evans, Chris, Chris Palmer, and Ryan Sleevi. *Public key pinning extension for HTTP*. No. RFC 7469. 2015.
[Eyal16] Eyal, Ittay, et al. "Bitcoin-NG: A scalable blockchain protocol." *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*. USENIX Association, 2016.