Contour: Practical Binary Transparency

On using Bitcoin’s consensus mechanism as a proactive transparency medium.

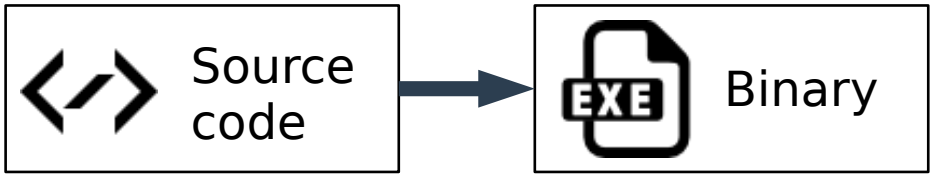
Mustafa Al-Bassam
@musalbas

What is binary transparency?

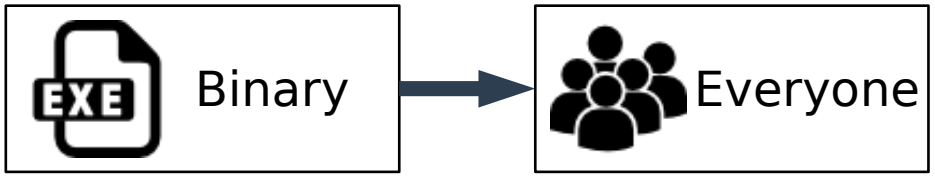
Knowing that the binary is you're being served is the same binary that everyone else is being served (or is at least public).

Completing the missing link

- **Reproducible builds: verifying that the source code matches the binary.**

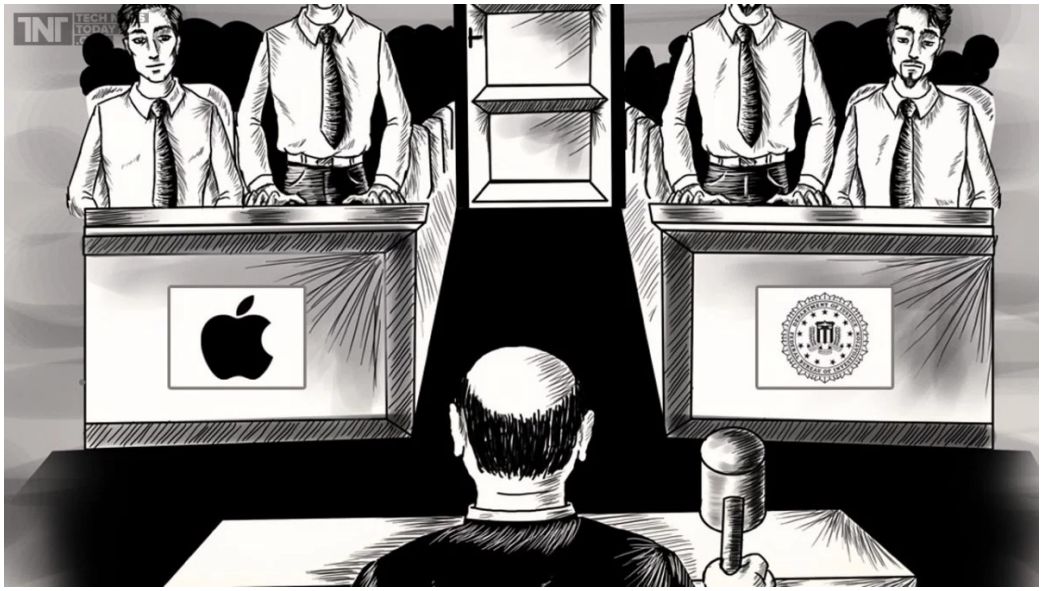


- **Binary transparency: verifying that the binary is known by all.**

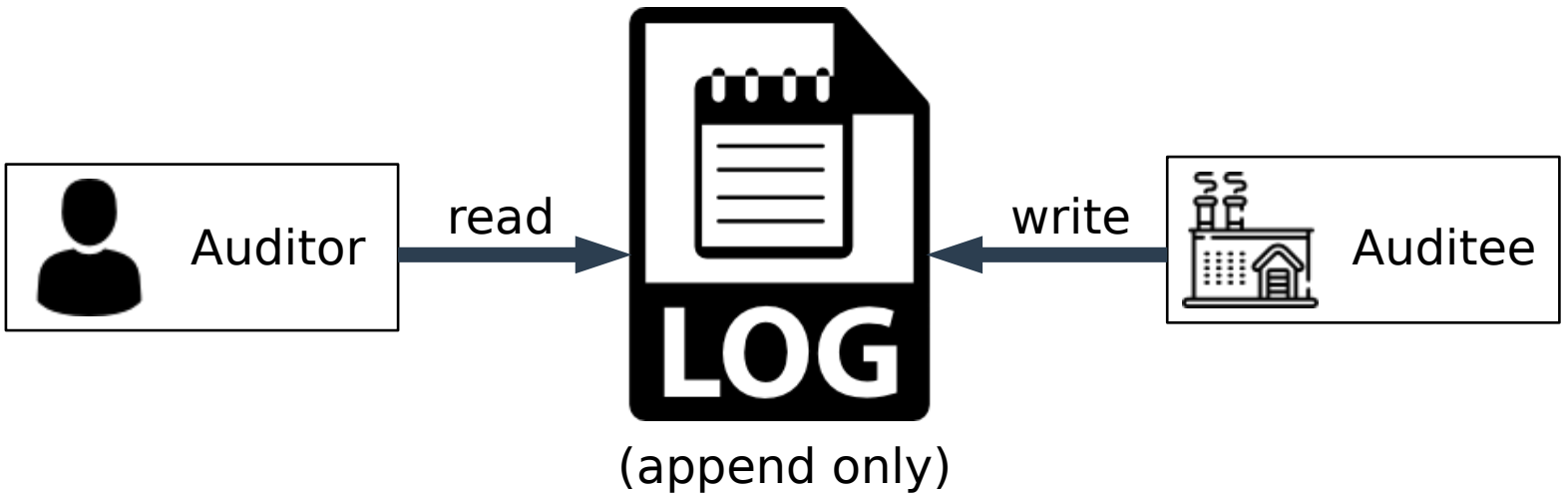


Why binary transparency?

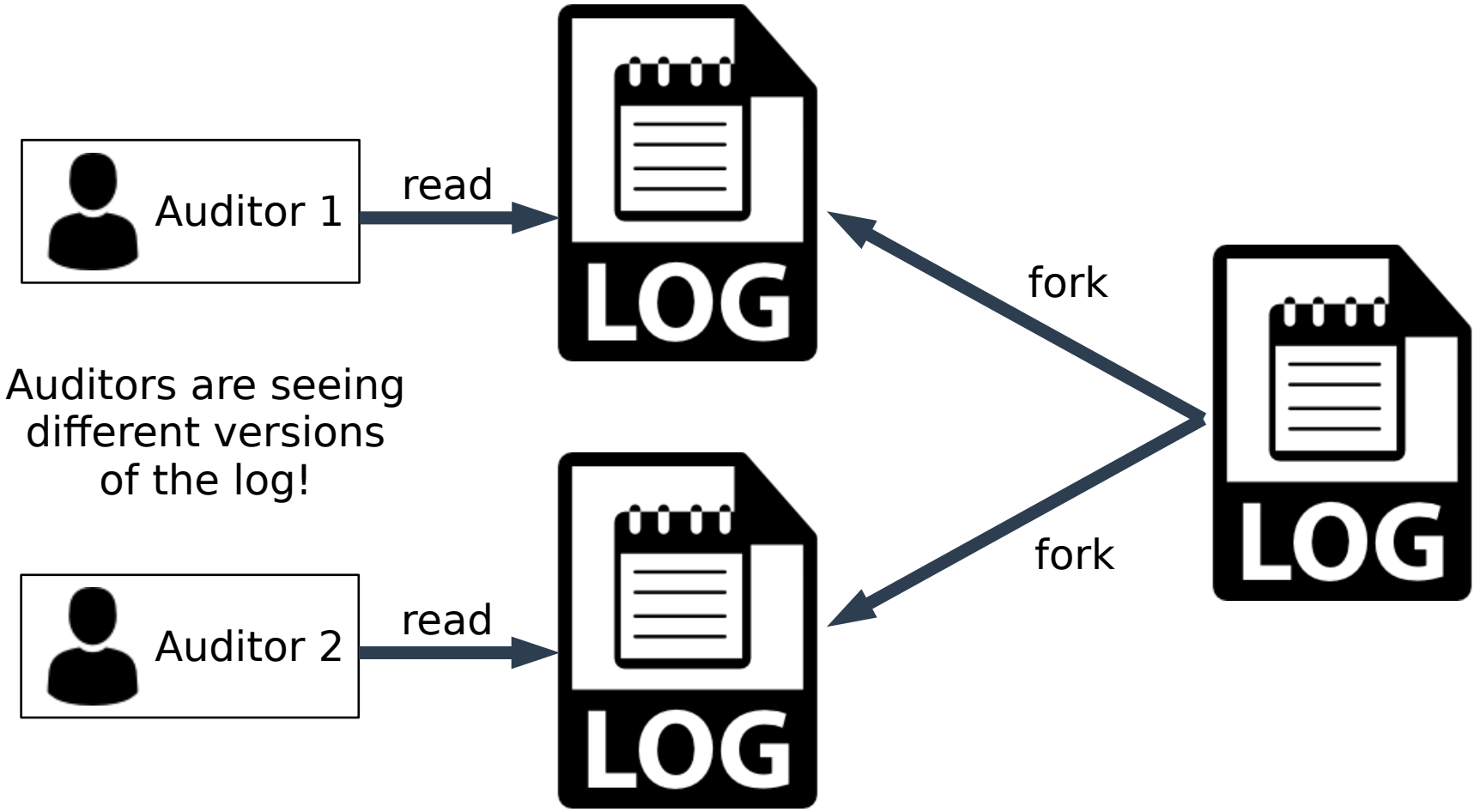
- **2012: NSA-linked “Flame” malware used rogue Microsoft binary signing certificate to infect users via Windows Update.**
- **2015: FBI ordered Apple in court to sign a backdoored version of iOS to bypass passcode rate limiting and unlock suspects’ phones.**



Basic model of transparency



Split view attacks (equivocation)



Ways of dealing with equivocation

Retroactive transparency
(e.g. Certificate Transparency)

&

Proactive transparency
(e.g. Bitcoin, BFT)

Retroactive transparency

- **Auditors gossip to each other about the log merkle roots – *after* they have acted on them - to see if they're being given the same version.**

- **This doesn't prevent log equivocation; it makes it *detectable*.**

Retroactive transparency is unsuitable for binary transparency

- **Gossiping isn’t yet practical.**
  - Google Chrome’s Certificate Transparency implementation still doesn’t have gossiping → Google’s log servers are a trusted third party!
  - It can be privacy invasive: log merkle roots can be used to fingerprint users.
- **Malicious binaries can disable gossip.**
- **Eclipsed or low-resource devices can be prevented from gossiping.**

Proactive transparency

- **Make equivocation *hard* to do in the first place, instead of just making it detectable.**
- **Basic example: a set of nodes have to achieve 'agreement' on every log update, so a majority would have to be dishonest for equivocation to happen (e.g. CoSi).**
  - You still need a censorship-resistant Sybil-resistance mechanism, or a governance mechanism, which is corruptible (look at EOS). :-(
- **Bitcoin achieves proactive transparency using *cryptoeconomics*: long-term equivocation is *expensive*.**
  - 51% attacks, eclipse attacks.

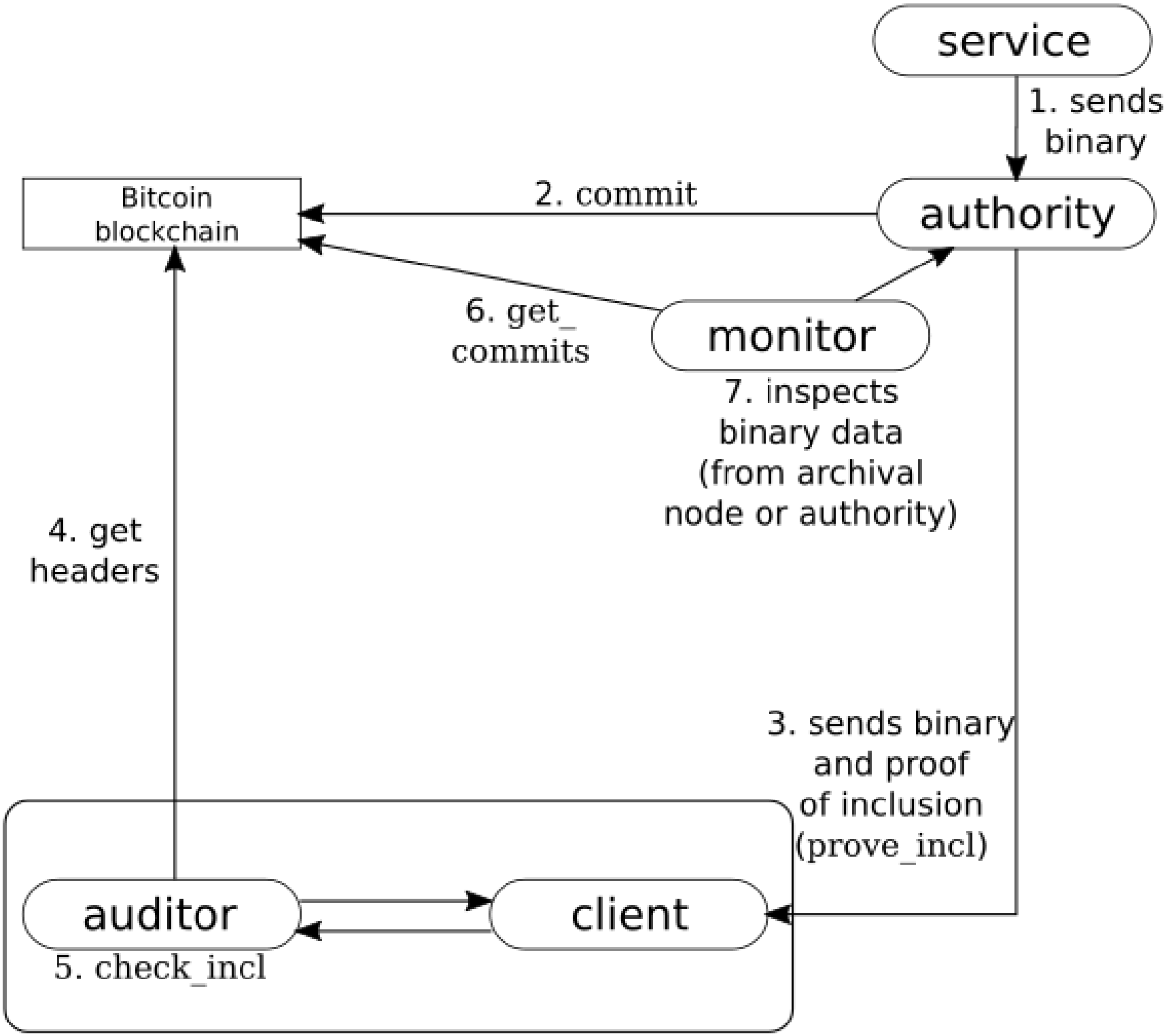
Threat model

Actors

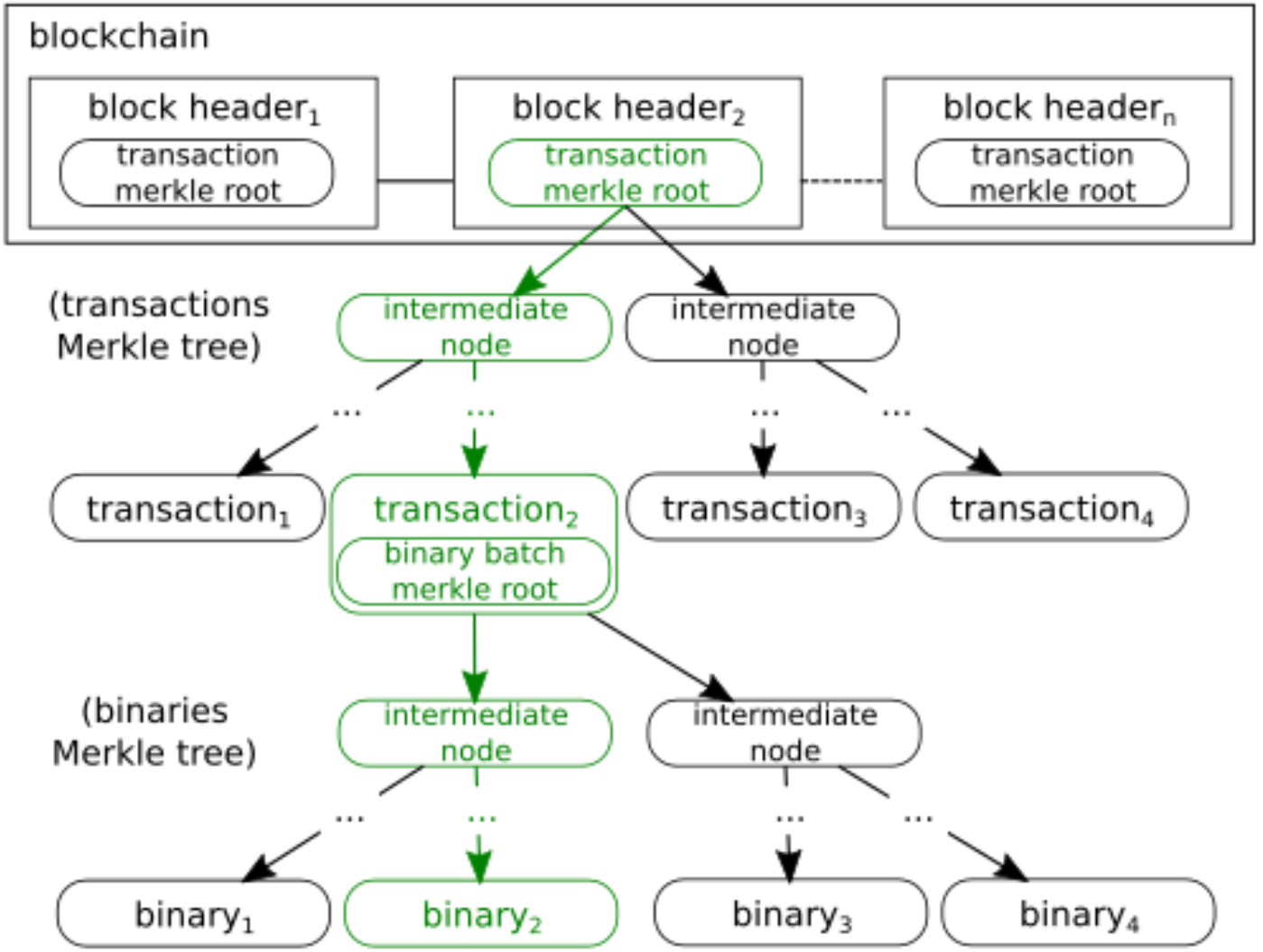
- **Service – issues software (i.e. developers).**
- **Authority – distributed software (i.e. app store). Has a known Bitcoin address.**
- **Monitor – inspects log.**
- **Auditor – checks that binaries are in log.**
- **User – receives updates, checks with auditor they're in the log.**

Assumptions

- **Authority is untrusted.**
- **Auditor/user may be compromised after acting on a log entry (i.e. installing update).**
- **Auditor/user's local network is untrusted (eclipse attacks!)**

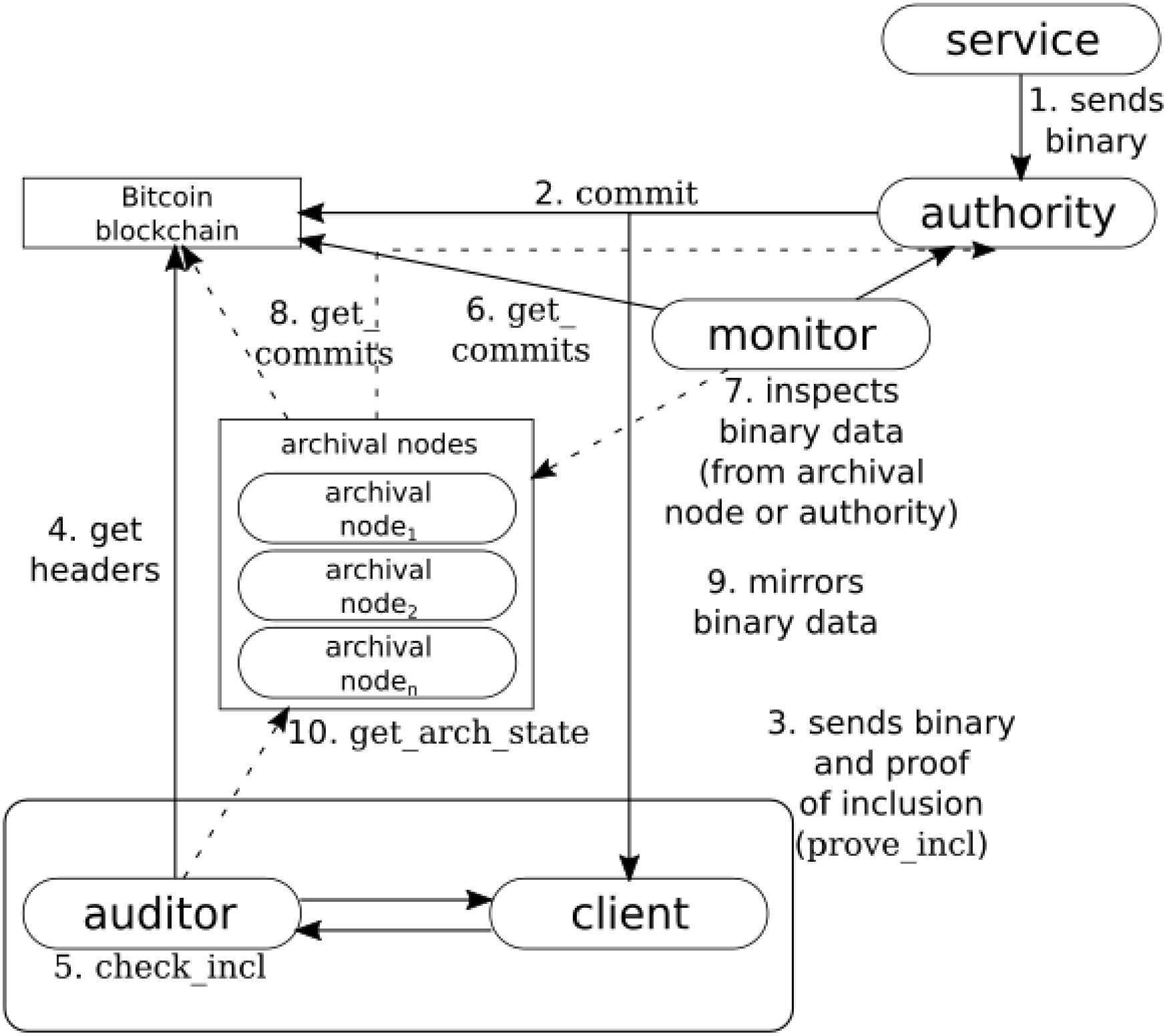

service
1. sends binary
Bitcoin blockchain
2. commit
authority
6. get_ commits
monitor
7. inspects binary data (from archival node or authority)
4. get headers
3. sends binary and proof of inclusion (prove_incl)
auditor
client
5. check_incl

Inclusion proof structure



What about data availability?

- **What if an authority publishes a merkle root to the blockchain, but does not actually publish the binaries anywhere?**
  - Misbehaviour will still be detected, as it will become known that no one has a copy of these binaries, which have been committed to on the blockchain.
- **But what if we want granular misbehaviour detection; i.e. we want to be able to inspect the actual binaries?**
  - We can use archival nodes, which attest that the data is available; in the context of Linux repos, these can be local mirrors. Auditors trust these nodes.


service
1. sends binary
Bitcoin blockchain
2. commit
authority
8. get_ commits
6. get_ commits
monitor
7. inspects binary data (from archival node or authority)
archival nodes
archival node1
archival node2
archival noden
4. get headers
9. mirrors binary data
10. get_arch_state
3. sends binary and proof of inclusion (prove_incl)
auditor
client
5. check_incl

Cost of peforming a split view attack (as of December 2017)

- **If you can’t do an eclipse attack**
  - You have to do a 51% attack.
  - Hardware cost: ~$2B using Antminer S9s
  - Electricity cost:~$120K per hour ($0.10 per kilowatt hour)
- **If you can do an eclipse attack**
  - If you want to do the attack within a week, and auditors require 6 confirmations:
  - Hardware cost: ~$8.3M
  - Electricity cost: ~$100k
  - Per device-block header.

Some performance numbers...

Operation	Time (µs)	σ (µs)
`commit`	**5.93 (s)**	**0.297 (s)**
`prove_incl` (one-time)	8.5	5.4
`prove_incl` (per statement)	12	6.4
`check_incl`	224	62.14

Table 4: Average time of individual operations, and standard deviation σ, when the batch size is 1M. The timings for `commit` were averaged over 20 runs, and for `prove_incl` and `check_incl` over 1M runs. The timings for `commit` are in bold to emphasize that they are in seconds, not microseconds.

Operation	Bandwidth
`Authority.commit` (using APIs)	1 MB
`Authority.commit` (one-time setup for full node)	126 GB
`Authority.commit` (using full node)	235 B
`Auditor.sync`	37.4 MB
`Auditor.prove_incl`	1.3 kB

Table 5: The bandwidth cost of operations, when the batch size is 1M. The cost of `Authority.commit` depends on whether or not the authority is running a full Bitcoin node or relying on third party APIs. For running a full node, there is a one-time setup cost to synchronize the blockchain.

What would it look like if you implemented this for Debian apt?

- **As of January 2017, Debian software repos contain ~1.7TB of data, with 1040 package updates daily.**
- **Updates 4 times a day → 4 Bitcoin transaction a day.**
- **~1.3KB maximum storage and bandwidth overhead per package, for storing and sending inclusion proofs.**
- **Users would need to run SPV clients, download ~39MB of block header, but only store the hashes, which is ~15.9MB of data**
  - 11.5KB/day to download; 5.6KB/day to store

Thank you

Questions?

github.com/musalbas/contour

Twitter: @musalbas
Email: mus@musalbas.com