Catena: Efficient Non-equivocation via bitcoin

Alin Tomescu
alinush@mit.edu

Srinivas Devadas
devadas@mit.edu

CSAIL

Overview

1. What?
2. How?
3. Why?

Overview

1. **What?**
2. How?
3. Why?

The equivocation problem

The equivocation problem

Non-equivocation: *"Saying the same thing to everybody."*

The equivocation problem

Non-equivocation: *"Saying the same thing to everybody."*

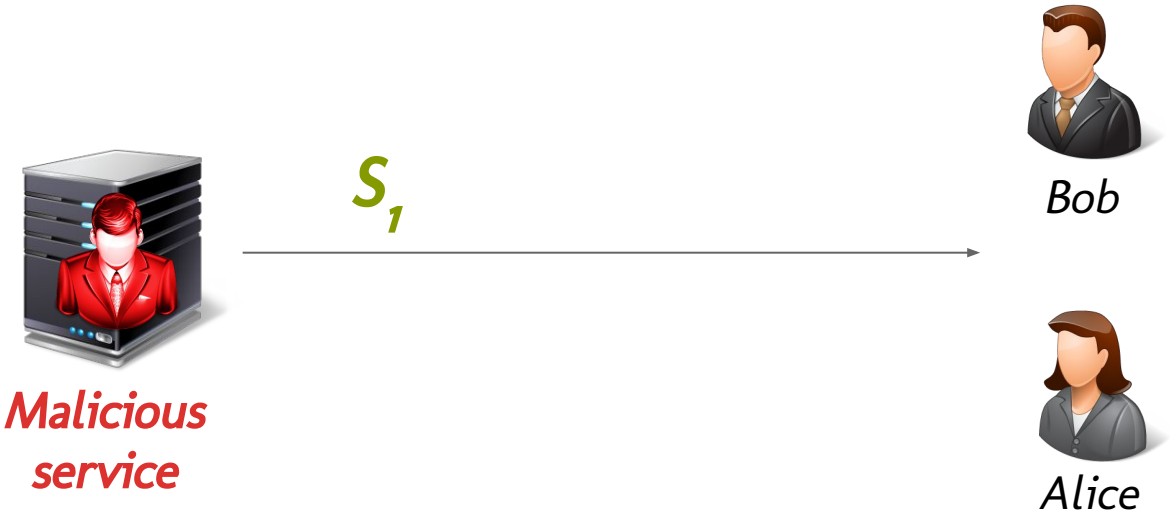
Malicious service

Bob

Alice

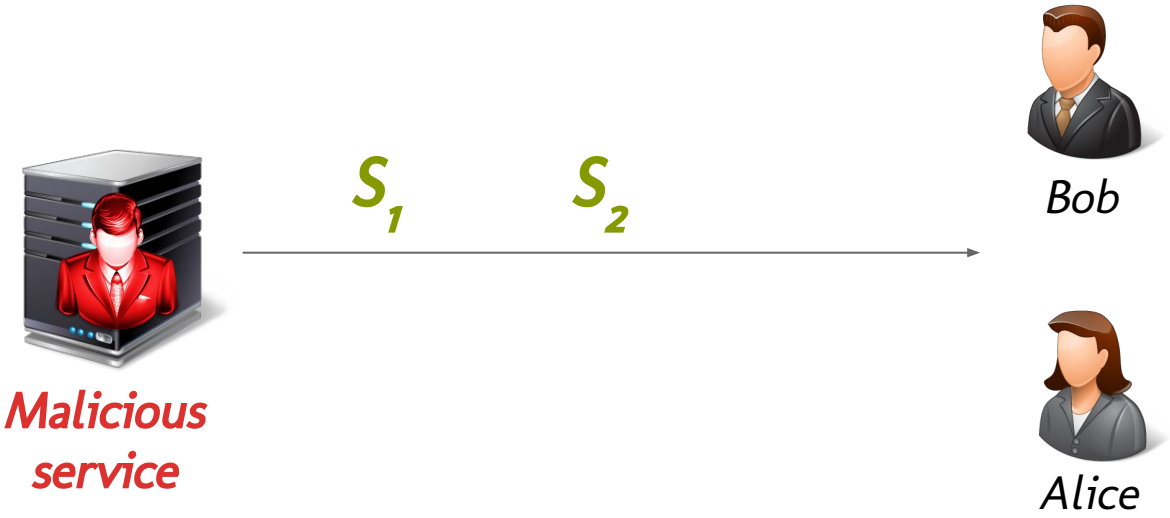
The equivocation problem

Non-equivocation: *"Saying the same thing to everybody."*



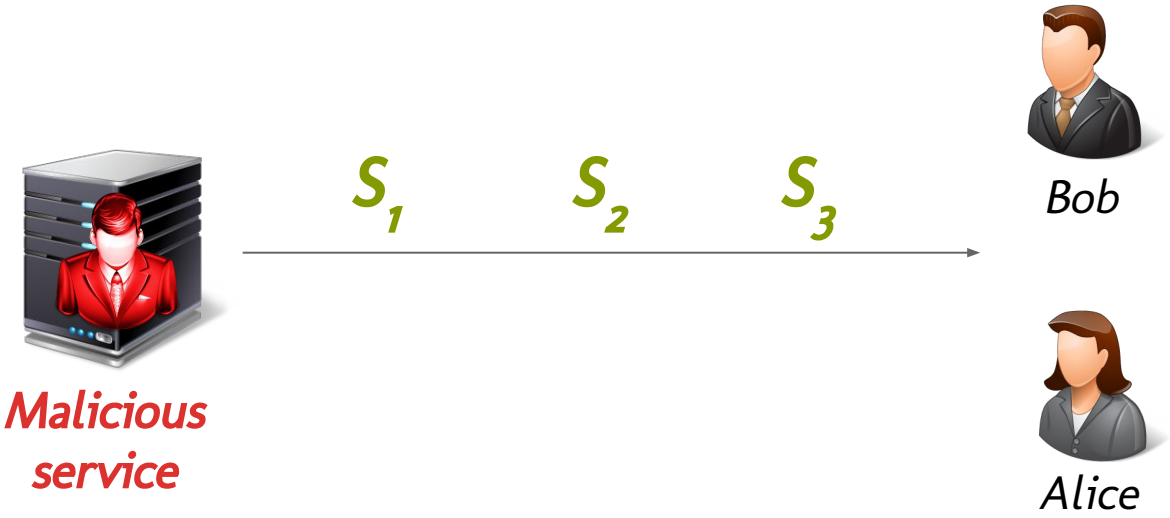
The equivocation problem

Non-equivocation: *"Saying the same thing to everybody."*



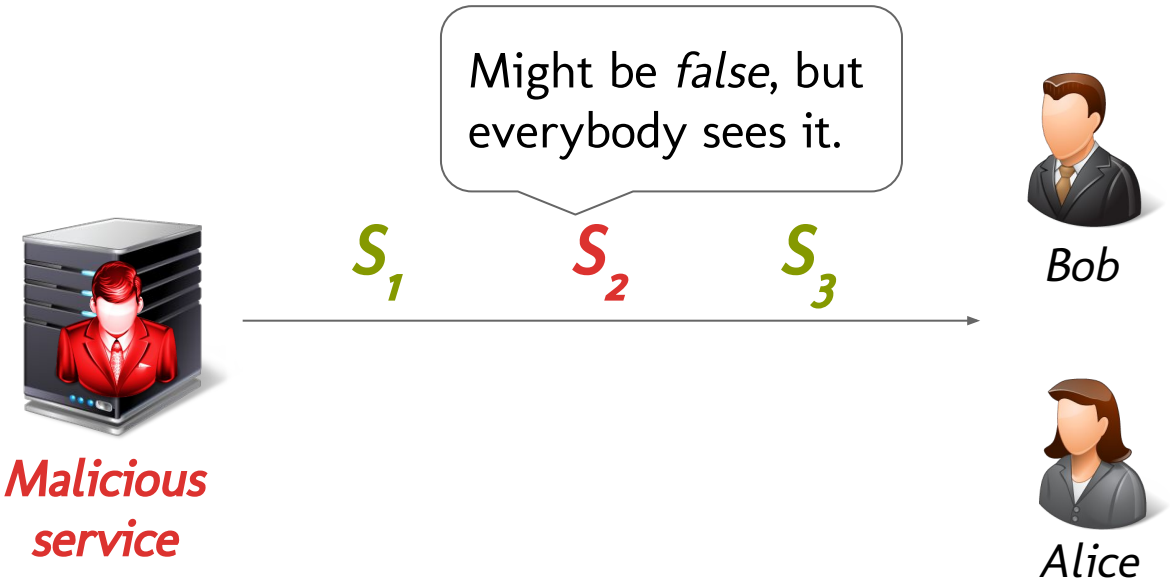
The equivocation problem

Non-equivocation: *"Saying the same thing to everybody."*



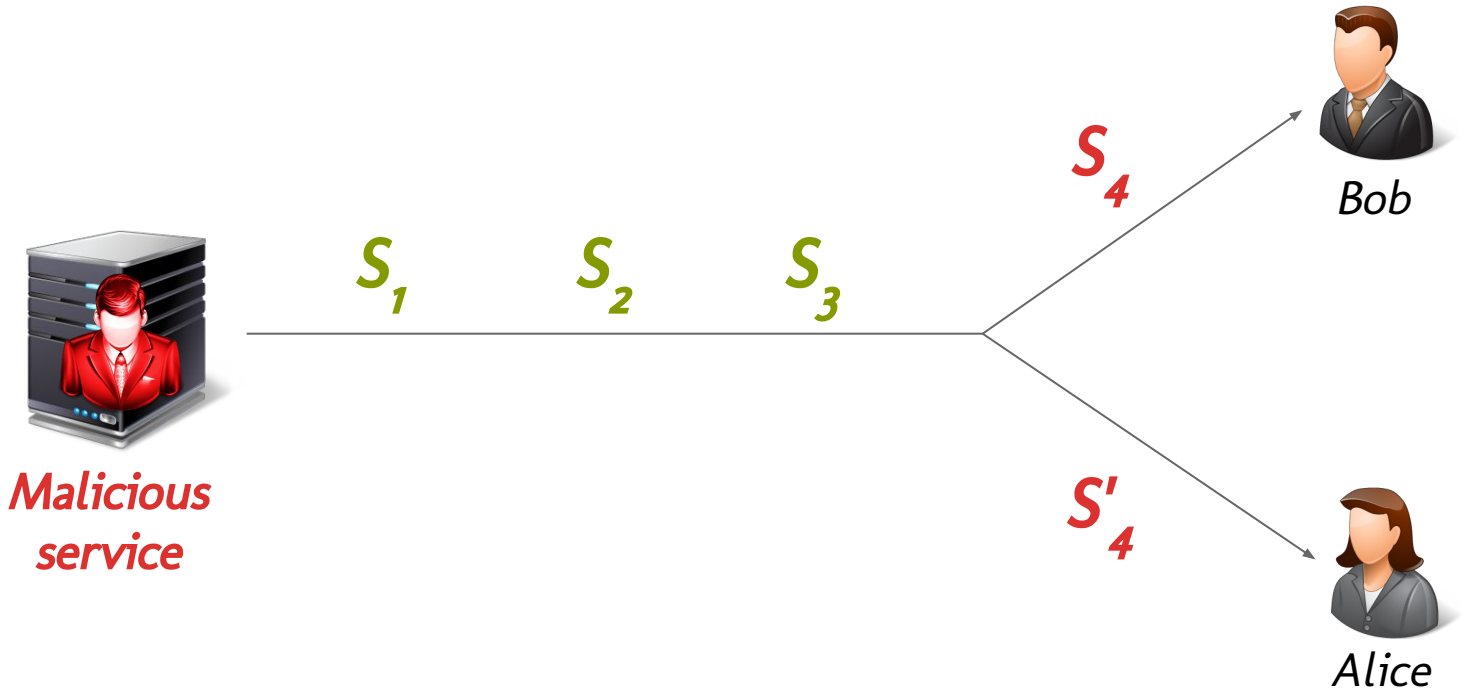
The equivocation problem

Non-equivocation: *"Saying the same thing to everybody."*



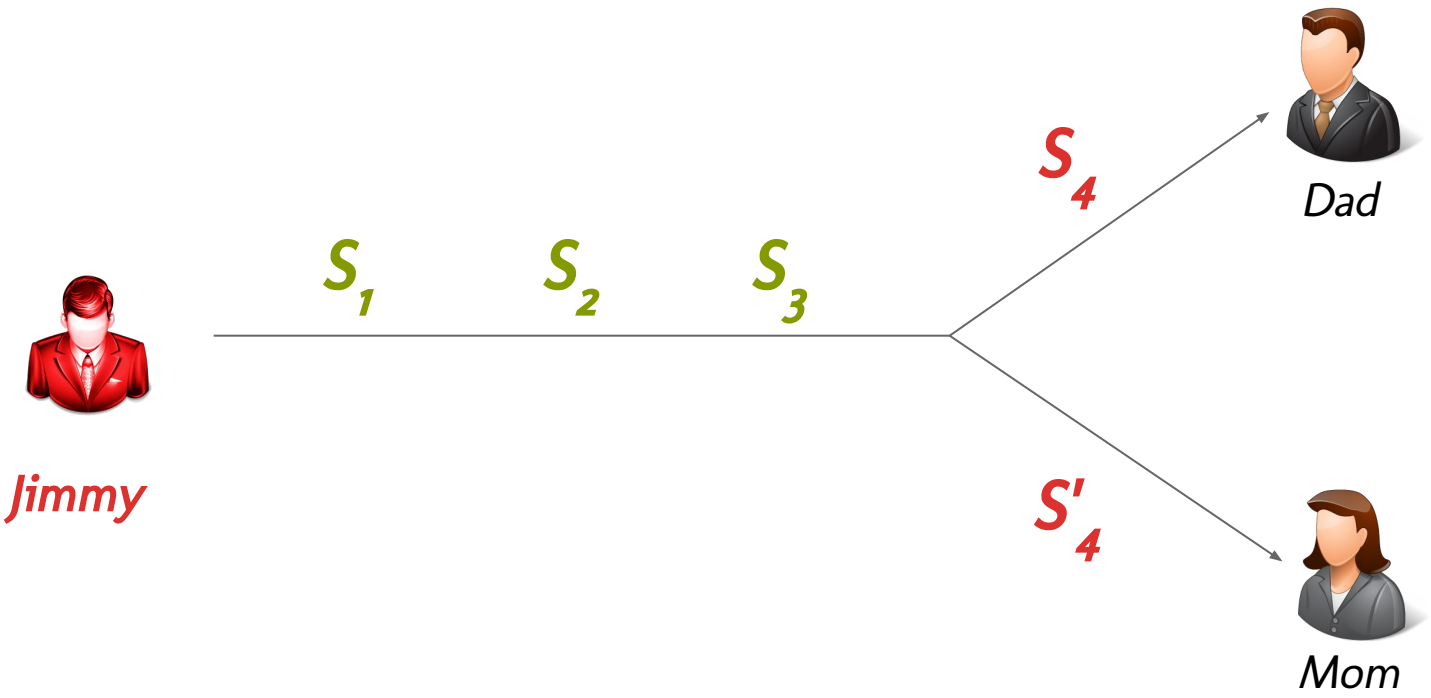
The equivocation problem

Equivocation: *"Saying different things to different people."*



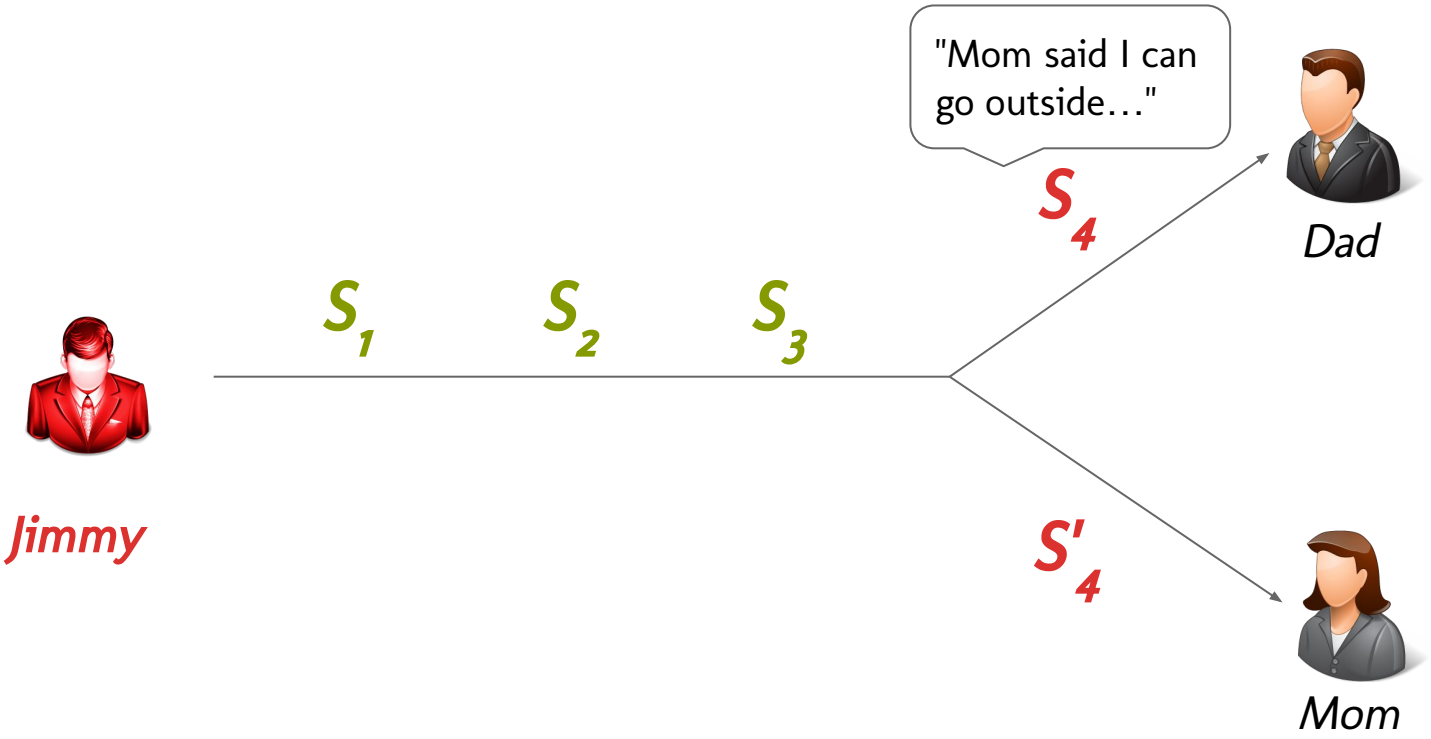
The equivocation problem

Equivocation: *"Saying different things to different people."*



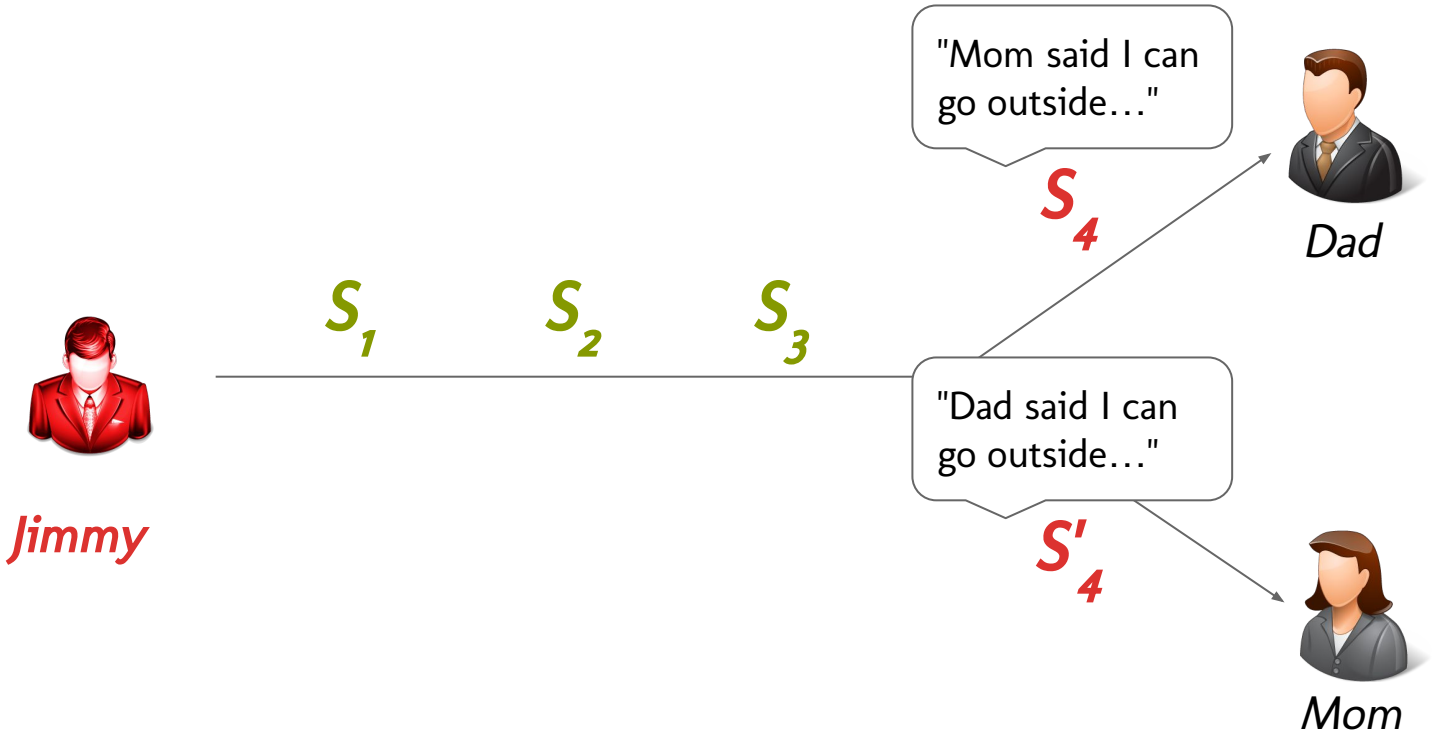
The equivocation problem

Equivocation: *"Saying different things to different people."*



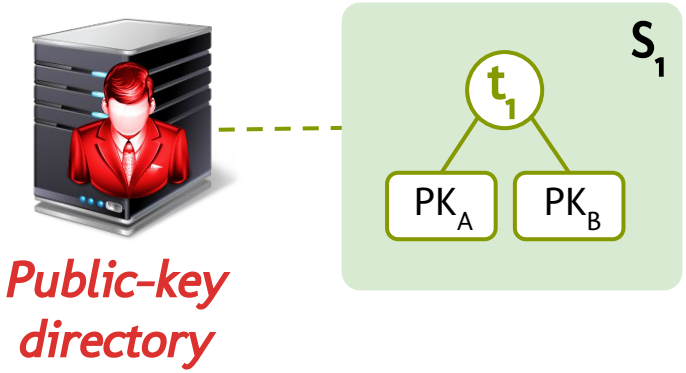
The equivocation problem

Equivocation: *"Saying different things to different people."*



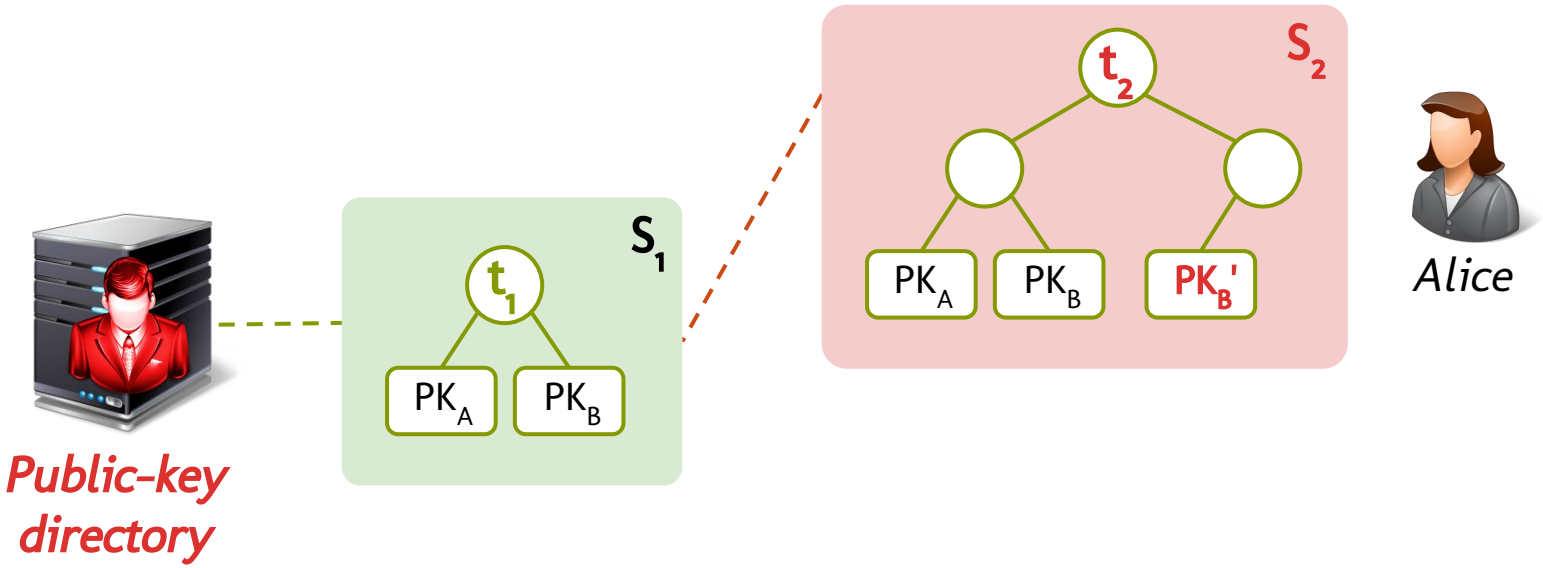
What is equivocation?

- At time **2**, malicious **server** publishes s_2 and s_2'



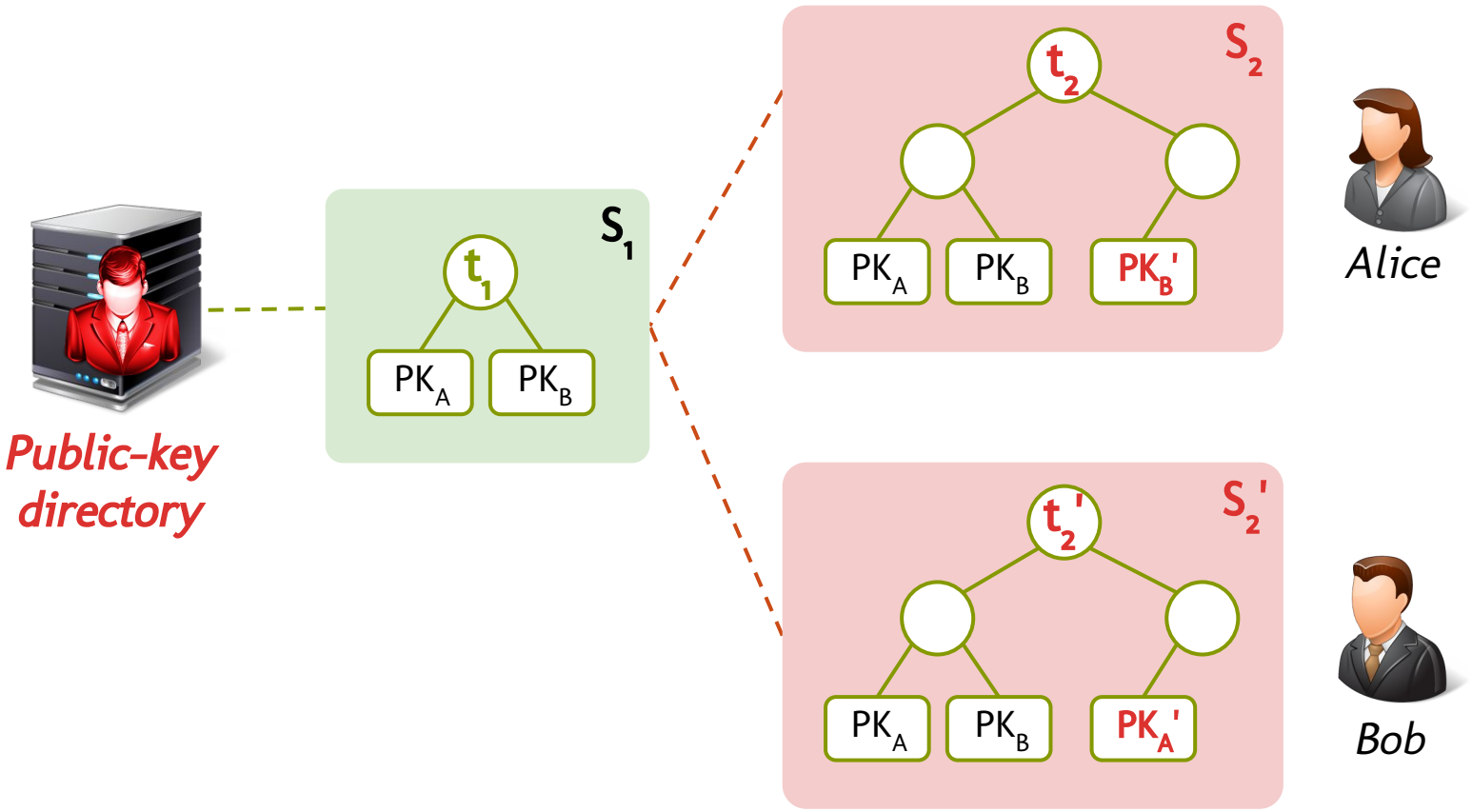
What is equivocation?

- s_2: Leave Alice's key intact, add fake PK_B' for Bob



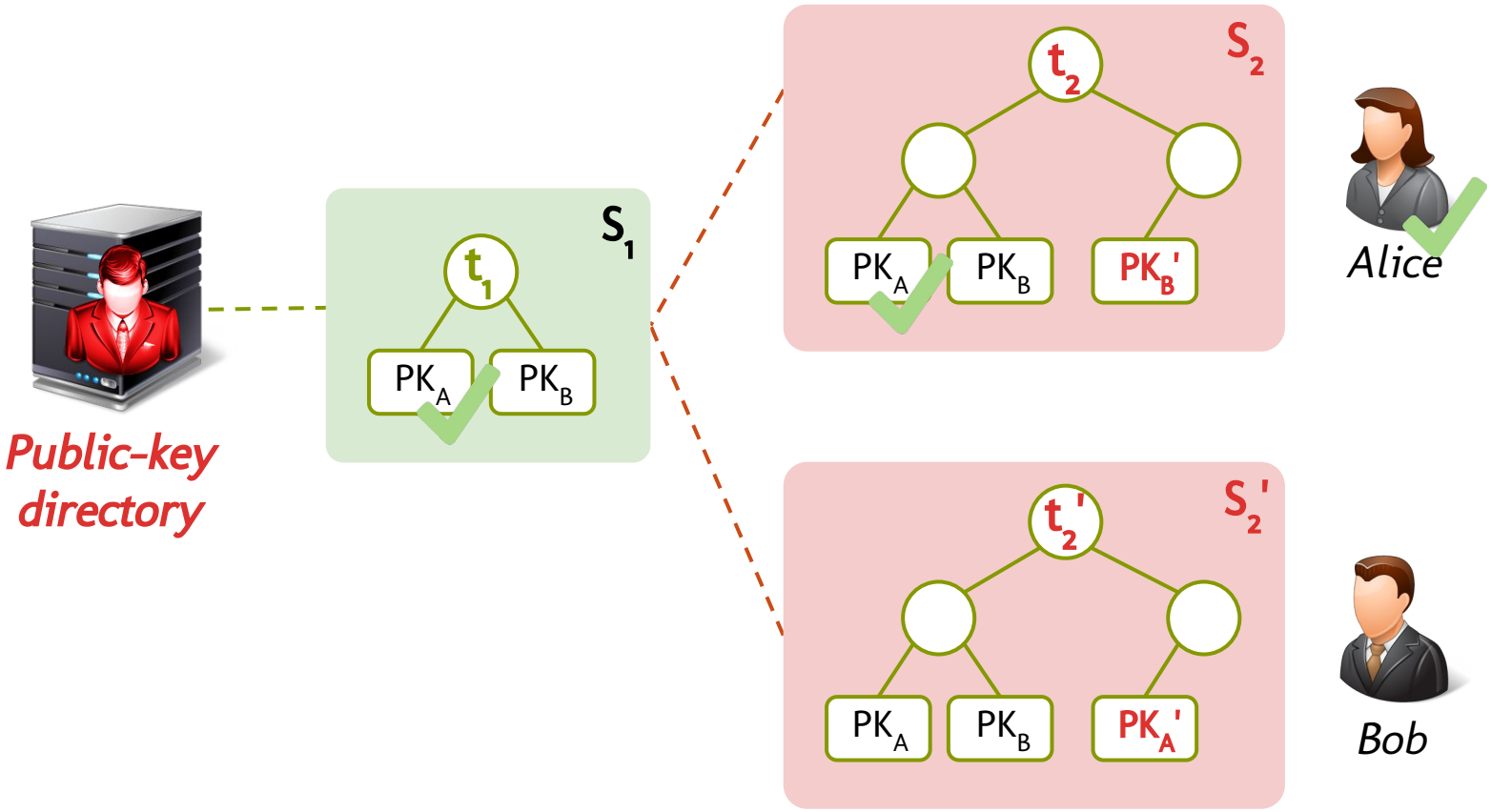
What is equivocation?

- s_2': Leave Bob's key intact, add fake PK_A' for Alice



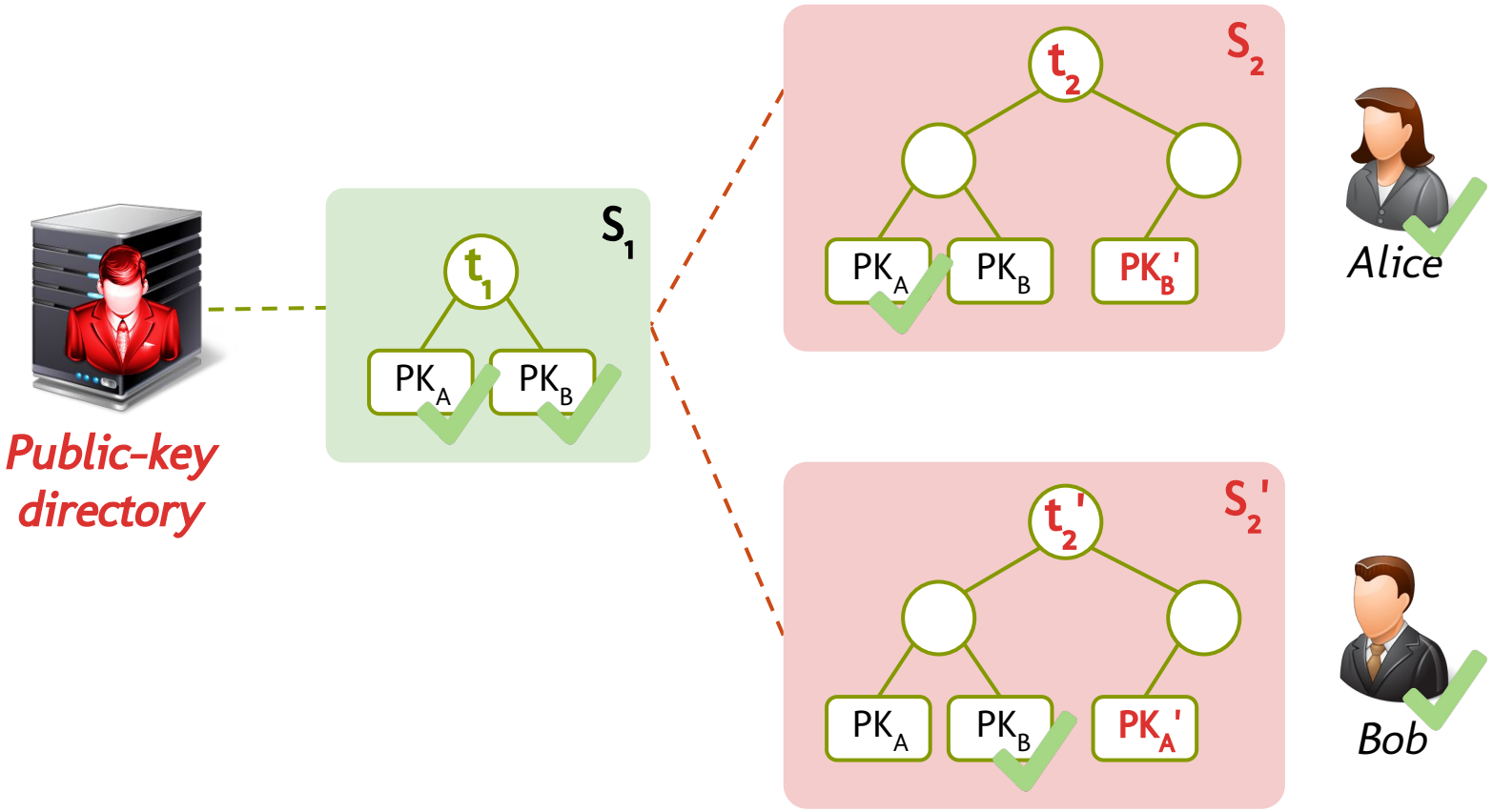
What is equivocation?

- Alice not impersonated in her view, but Bob is.



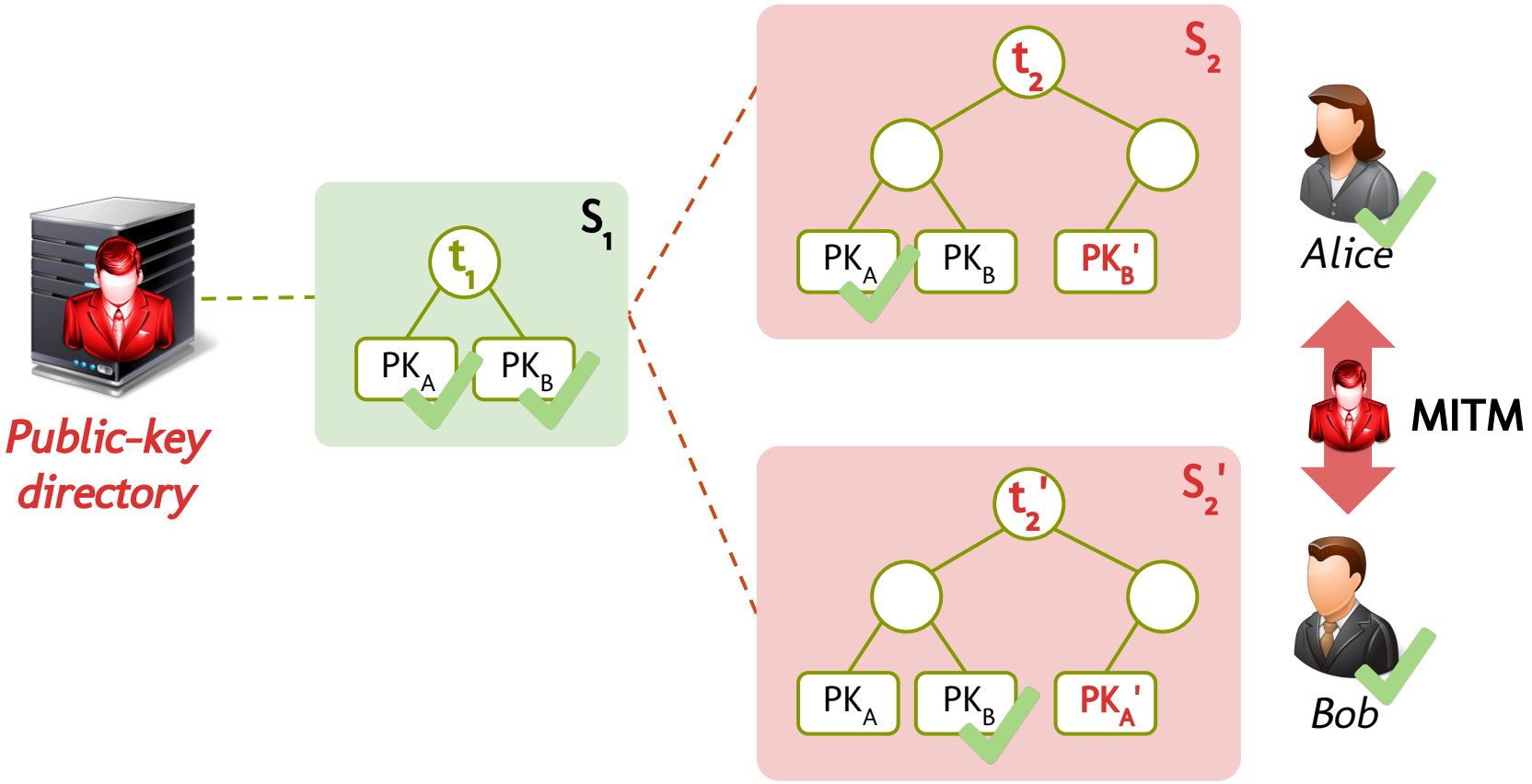
What is equivocation?

- Bob not impersonated in his view, but Alice is.



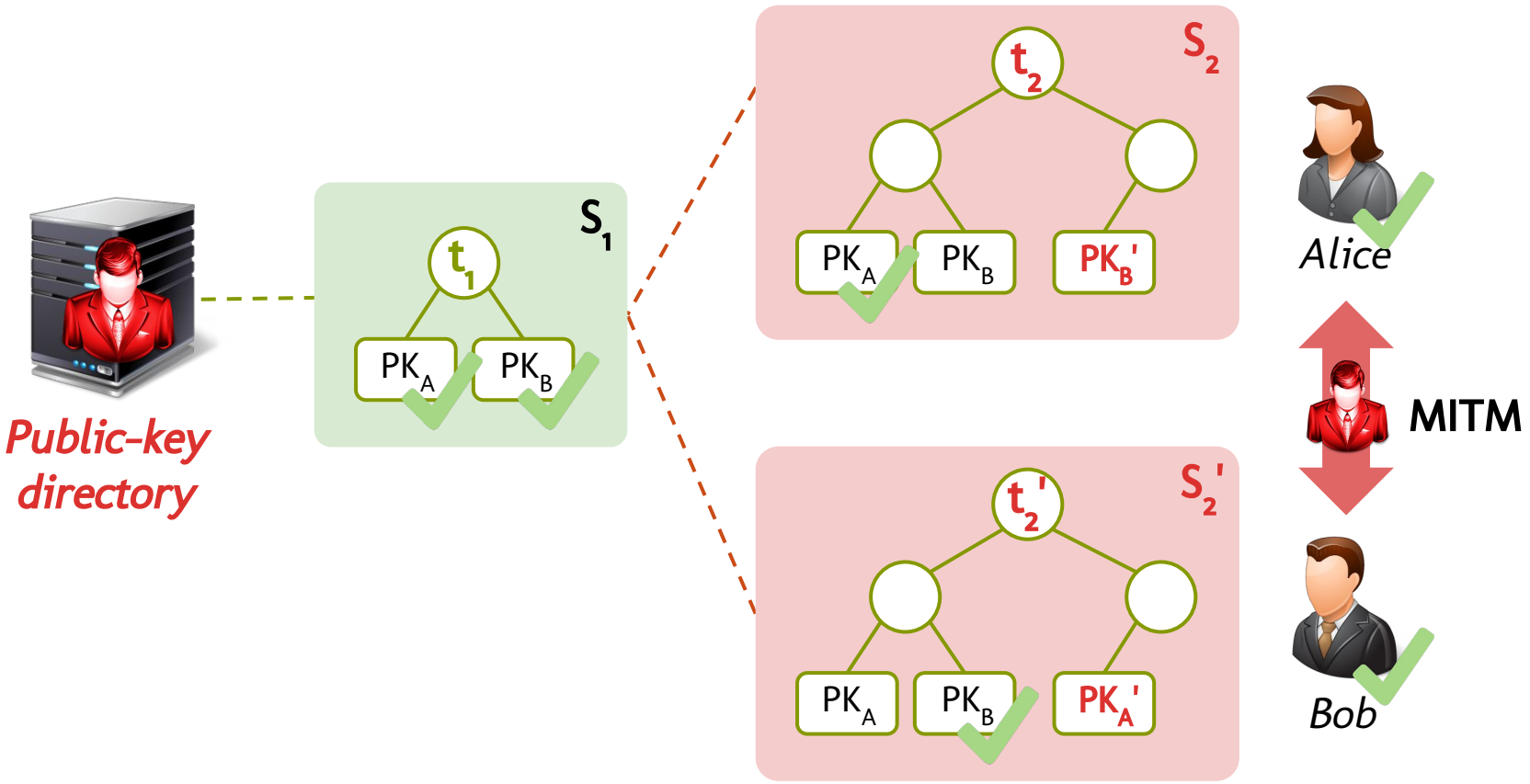
What is equivocation?

- Obtain fake keys for each other ⇒ MITM



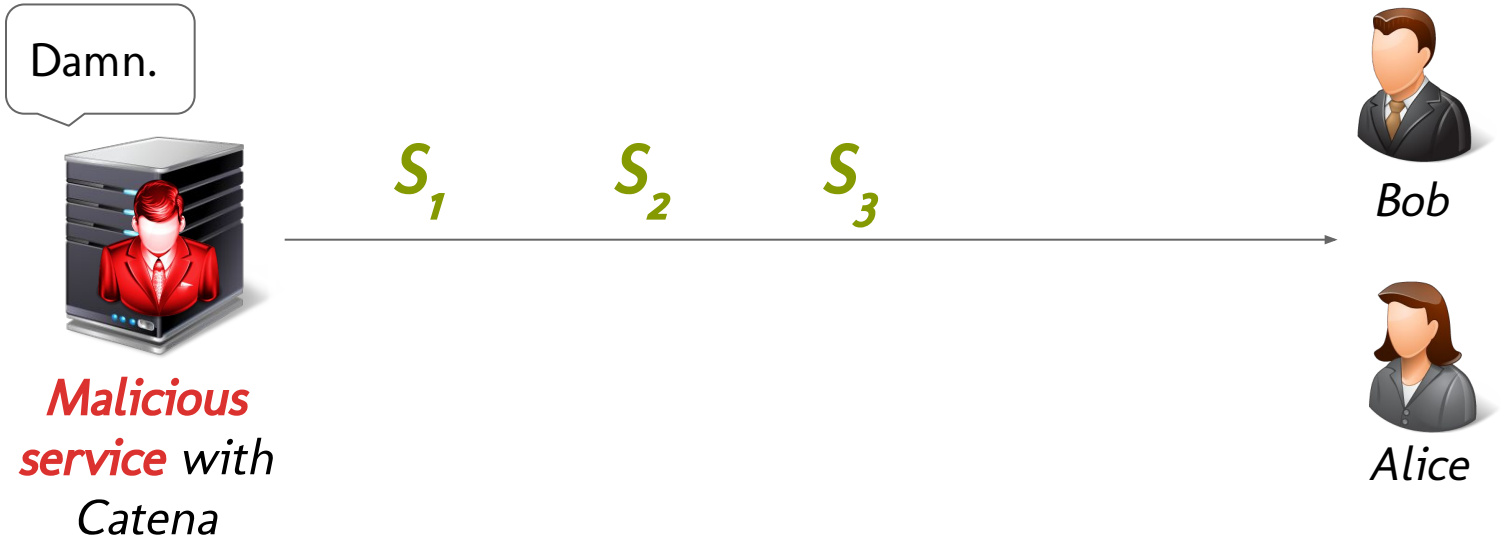
What is equivocation?

Bad: *"Stating different things to different people."*

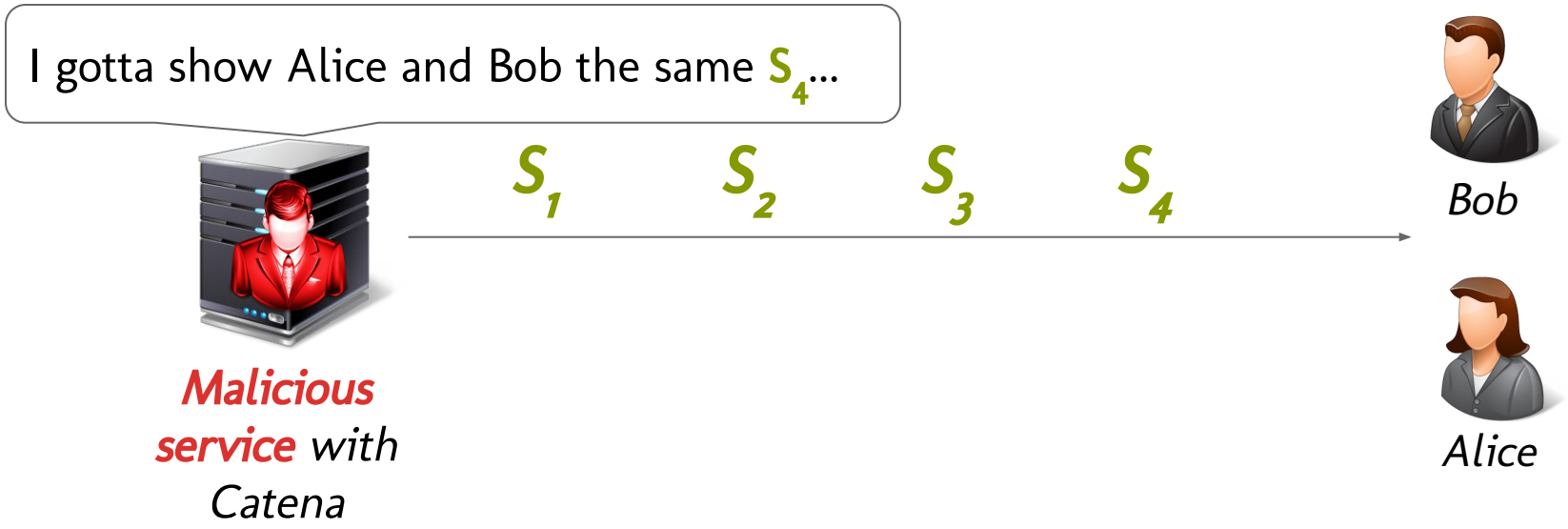


Catena prevents equivocation!

Catena prevents equivocation!



Catena prevents equivocation!



So what?

So what?

Secure software update

So what?

Secure software update

- Attacks on Bitcoin binaries

0.13.0 Binary Safety Warning

17 August 2016

Summary

Bitcoin.org has reason to suspect that the binaries for the upcoming Bitcoin Core release will likely be targeted by state sponsored attackers. As a website, Bitcoin.org does not have the technical resources to guarantee that we can defend ourselves from attackers of this calibre. We ask the Bitcoin community, and in particular the Chinese Bitcoin community to be extra vigilant when downloading binaries from our website.

So what?

Secure software update

- Attacks on Bitcoin binaries

0.13.0 Binary Safety Warning

17 August 2016

Summary

Bitcoin.org has reason to suspect that the binaries for the upcoming Bitcoin Core release will likely be targeted by state sponsored attackers. As a website, Bitcoin.org does not have the technical resources to guarantee that we can defend ourselves from attackers of this calibre. We ask the Bitcoin community, and in particular the Chinese Bitcoin community to be extra vigilant when downloading binaries from our website.

Secure messaging

- HTTPS
- *"We assume a PKI."*

So what?

Secure software update

- Attacks on Bitcoin binaries

0.13.0 Binary Safety Warning

17 August 2016

Summary

Bitcoin.org has reason to suspect that the binaries for the upcoming Bitcoin Core release will likely be targeted by state sponsored attackers. As a website, Bitcoin.org does not have the technical resources to guarantee that we can defend ourselves from attackers of this calibre. We ask the Bitcoin community, and in particular the Chinese Bitcoin community to be extra vigilant when downloading binaries from our website.

Secure messaging

- HTTPS
- *"We assume a PKI."*

"Blockchain" for X

10,000 feet view

10,000 feet view

- Bitcoin-based append-only log,

10,000 feet view

- Bitcoin-based append-only log,
    - Generalizes to other cryptocurrencies

10,000 feet view

- Bitcoin-based append-only log,
  - Generalizes to other cryptocurrencies
- ...as hard-to-fork as the Bitcoin blockchain
  - Want to fork? Do some work!

10,000 feet view

- Bitcoin-based append-only log,
  - Generalizes to other cryptocurrencies
- ...as hard-to-fork as the Bitcoin blockchain
  - Want to fork? Do some work!
- ...but efficiently auditable

10,000 feet view

- Bitcoin-based append-only log,
  - Generalizes to other cryptocurrencies
- ...as hard-to-fork as the Bitcoin blockchain
  - Want to fork? Do some work!
- ...but efficiently auditable
  - 600 bytes / statement (but can batch!)
  - 80 bytes / Bitcoin block

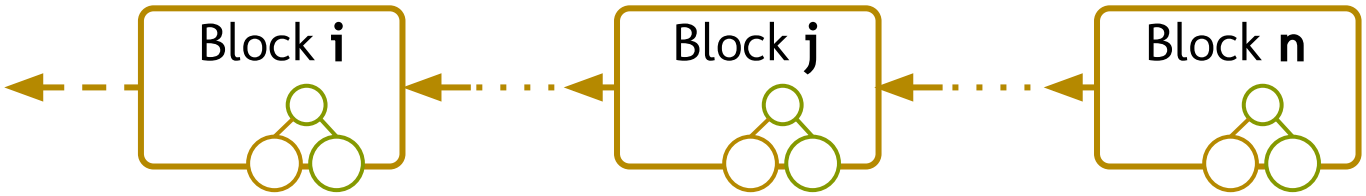
10,000 feet view

- Bitcoin-based append-only log,
  - Generalizes to other cryptocurrencies
- ...as hard-to-fork as the Bitcoin blockchain
  - Want to fork? Do some work!
- ...but efficiently auditable
  - 600 bytes / statement (but can batch!)
  - 80 bytes / Bitcoin block
- Java implementation (3500 SLOC)
  - https://github.com/alinush/catena-java

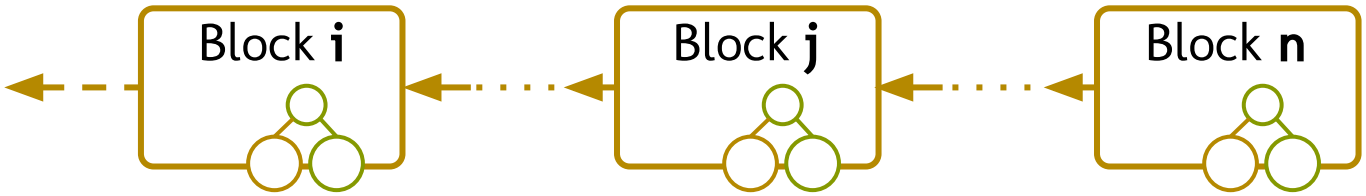
Overview

1. What?
2. How?
   a. **Bitcoin background**
3. Why?

Bitcoin blockchain

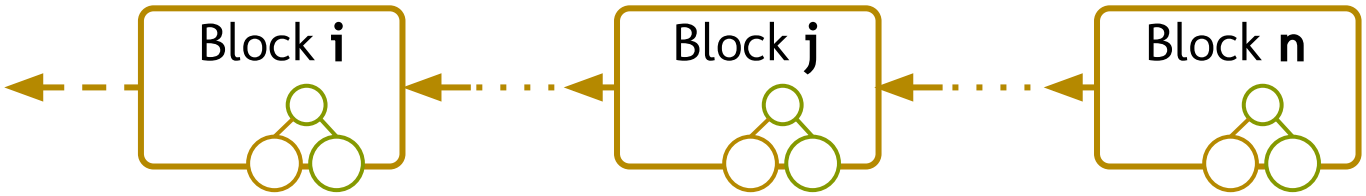


Bitcoin blockchain



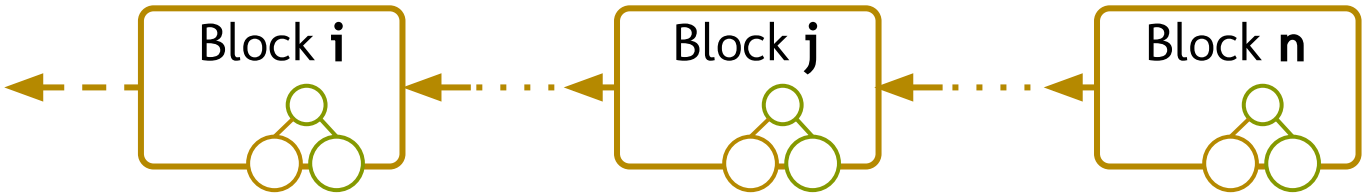
- Hash chain of blocks

Bitcoin blockchain



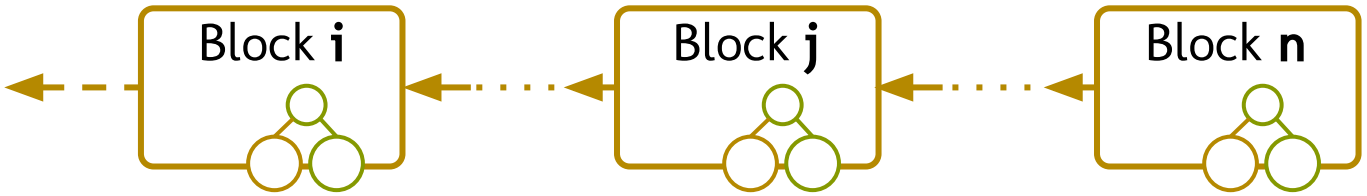
- Hash chain of blocks
  - Arrows are *hash pointers*

Bitcoin blockchain



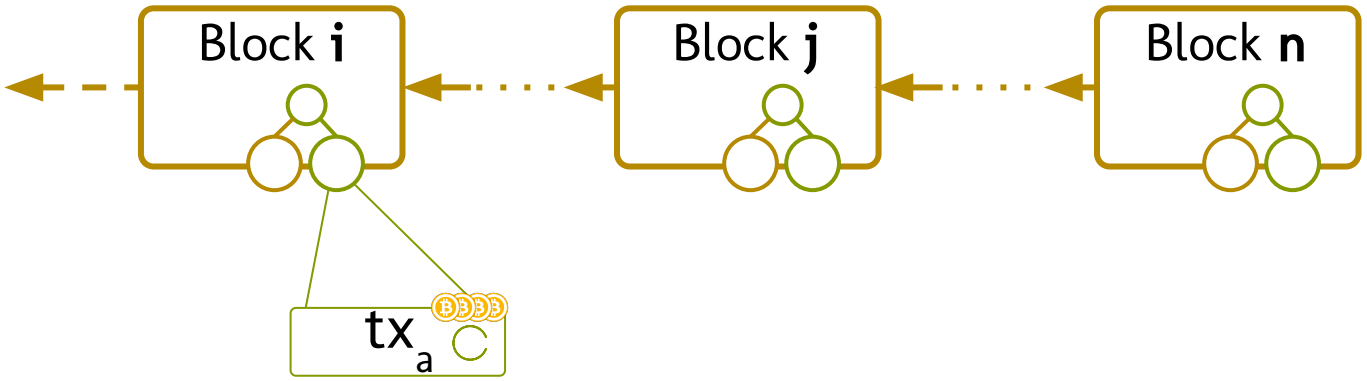
- Hash chain of blocks
  - Arrows are *hash pointers*
- Merkle tree of TXNs in each block

Bitcoin blockchain



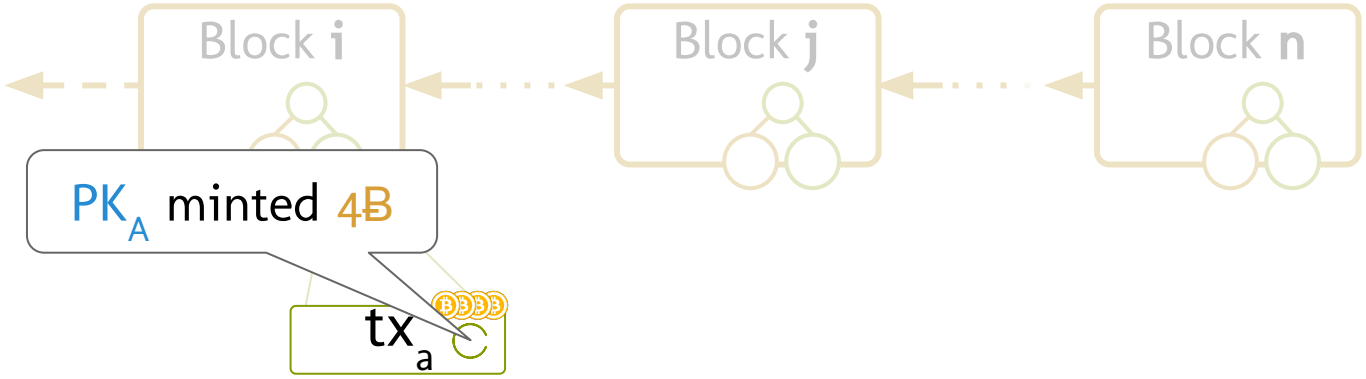
- Hash chain of blocks
  - Arrows are *hash pointers*
- Merkle tree of TXNs in each block
- Proof-of-work (PoW) consensus

Bitcoin blockchain



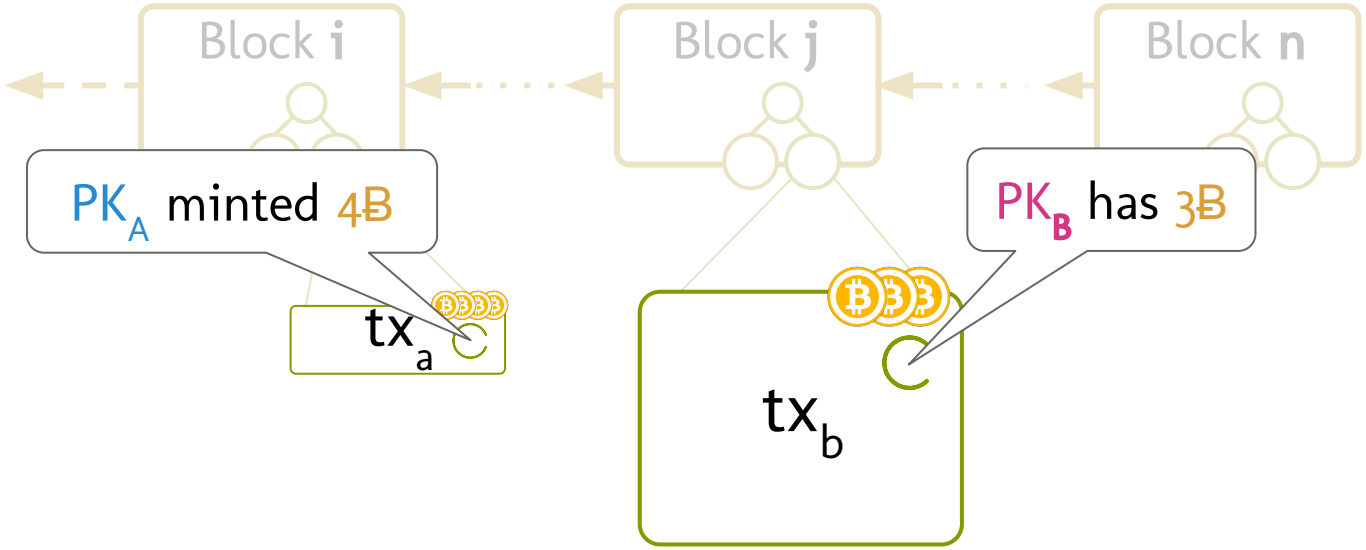
- Transactions mint coins

Bitcoin blockchain



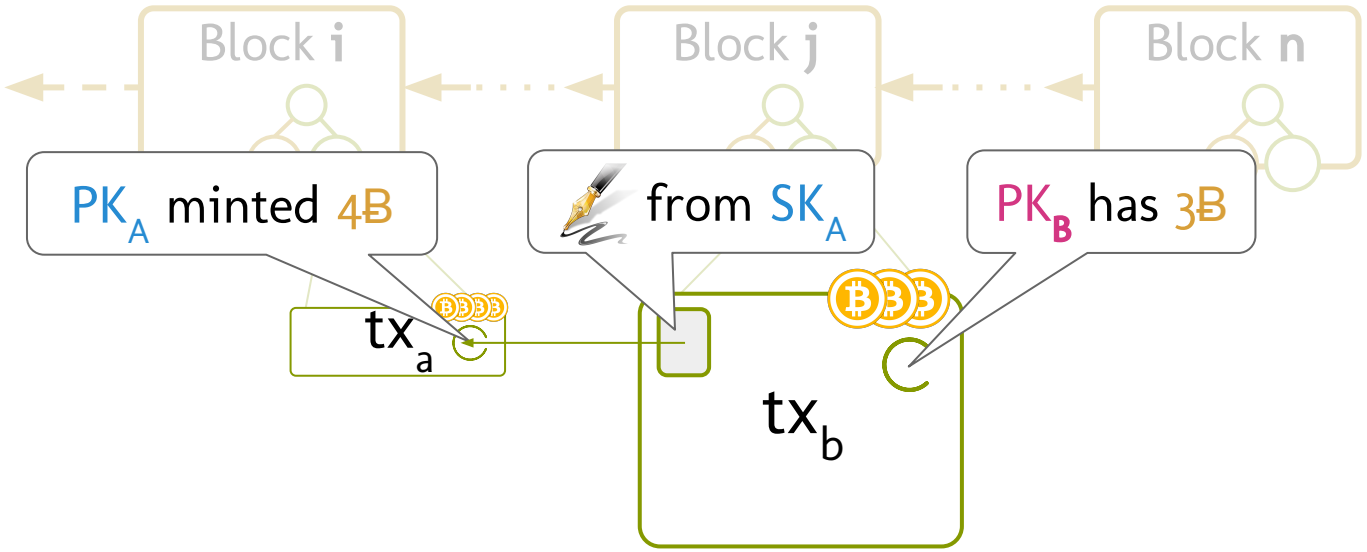
- Transactions mint coins
- Output = # of coins and owner's PK

Bitcoin blockchain



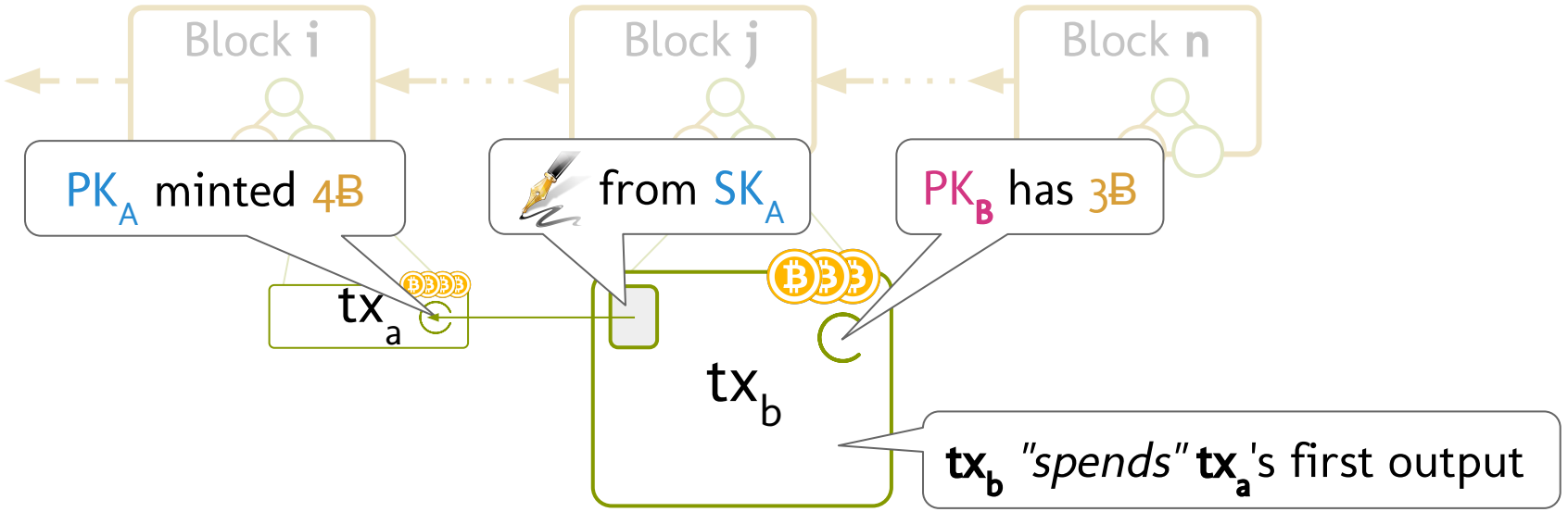
- Transactions mint coins
- Output = # of coins and owner's PK
- Transactions transfer coins (and pay fees)

Bitcoin blockchain



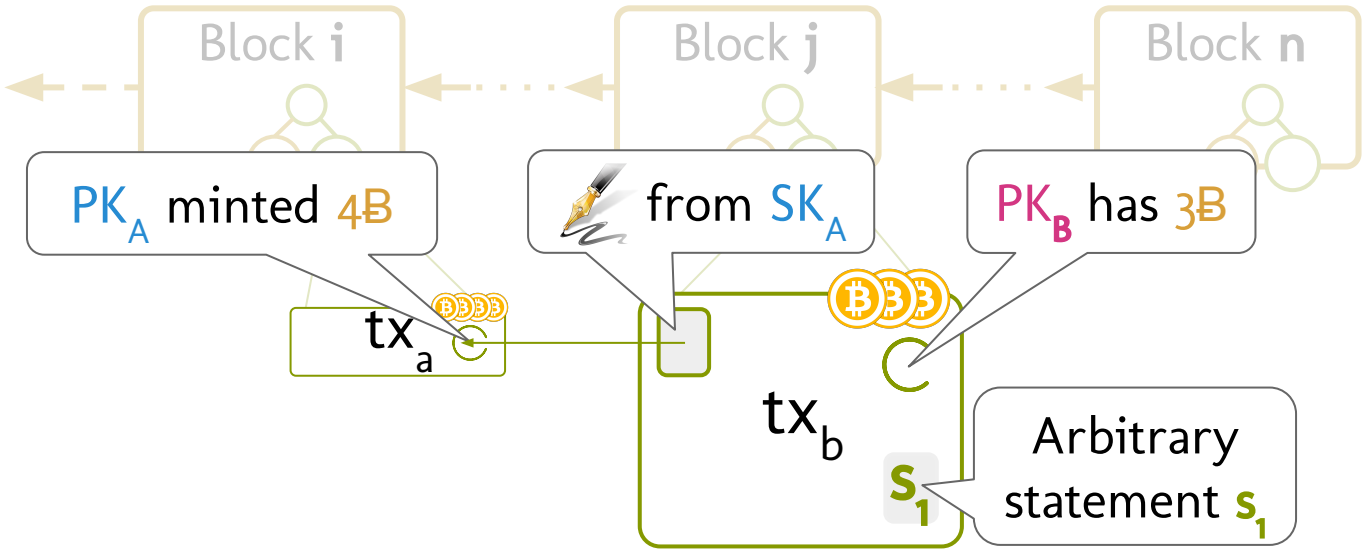
- Transactions mint coins
- Output = # of coins and owner's PK
- Transactions transfer coins (and pay fees)
- Input = hash pointer to output + digital signature

Bitcoin blockchain



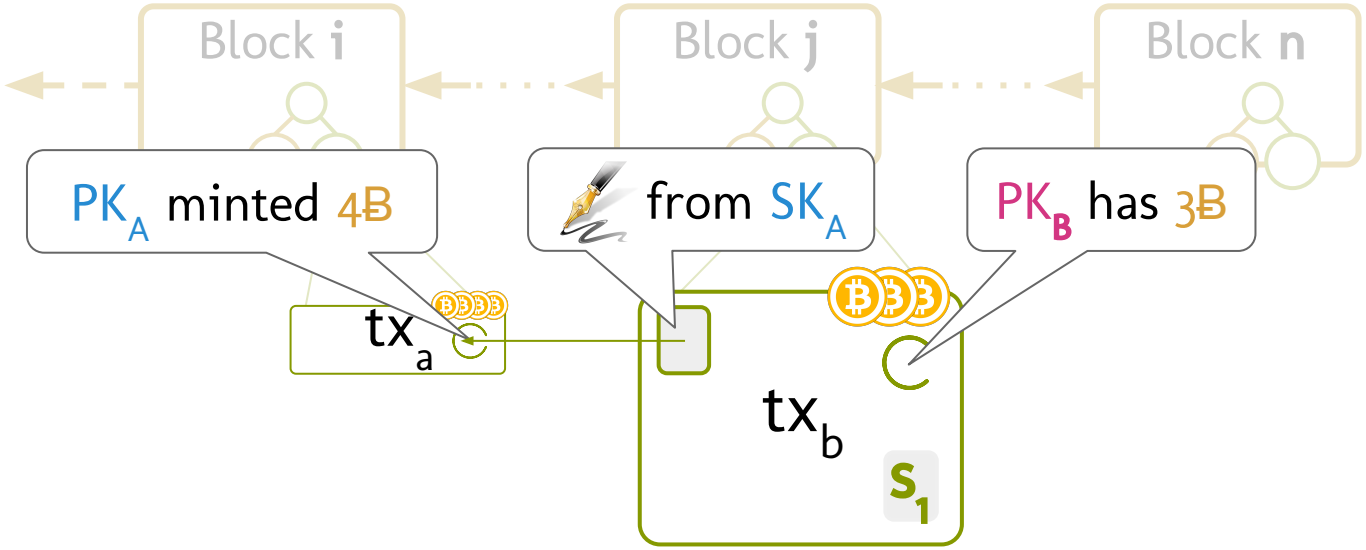
- Transactions mint coins
- Output = # of coins and owner's PK
- Transactions transfer coins (and pay fees)
- Input = hash pointer to output + digital signature

Bitcoin blockchain



Data can be embedded in TXNs.

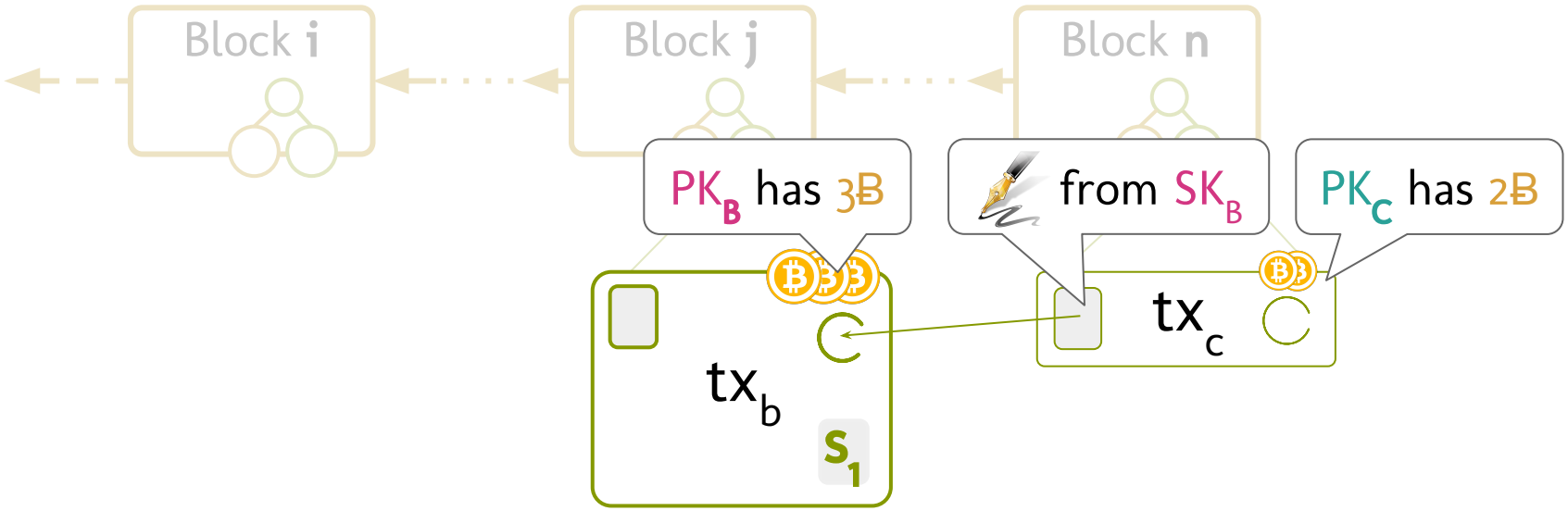
Bitcoin blockchain



Alice gives Bob 3Ƀ,

Bitcoin *miners* collected 1Ƀ as a *fee.*

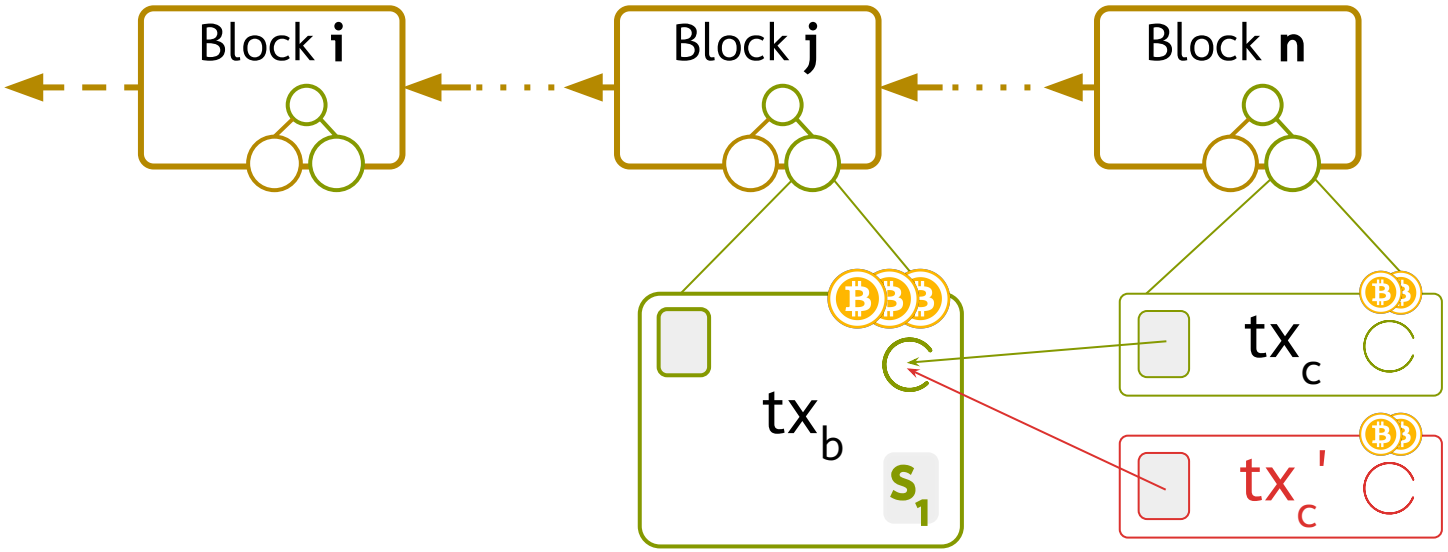
Bitcoin blockchain



Bob gives Carol 2Ƀ,

Bitcoin *miners* collected another Ƀ as a fee.

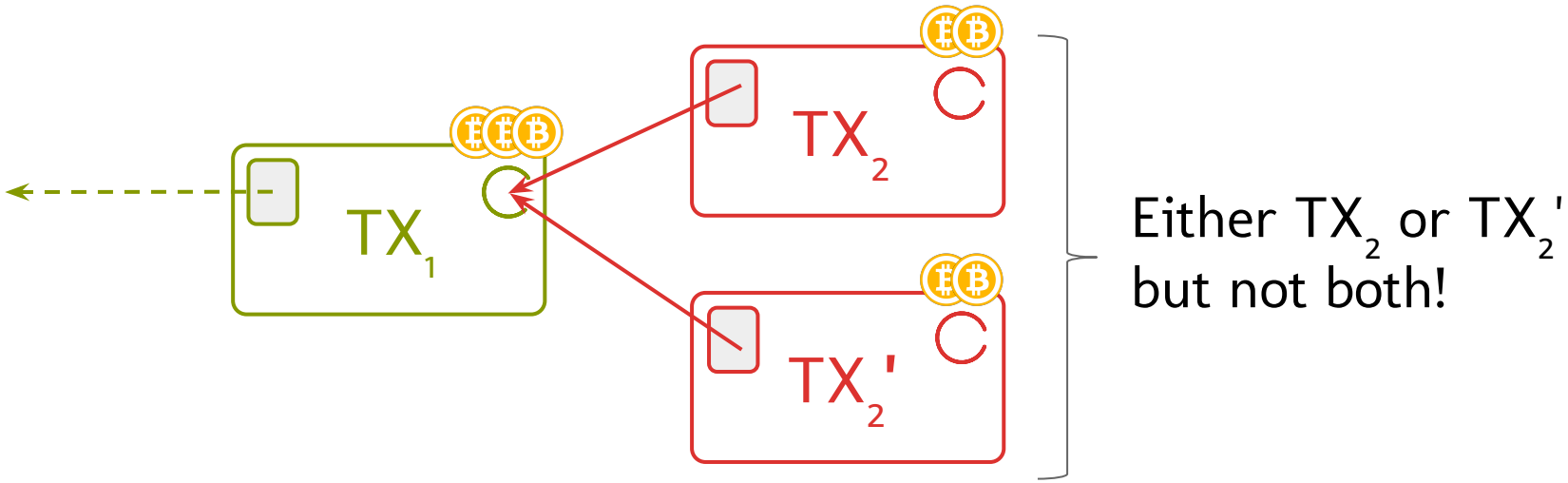
Bitcoin blockchain



No double-spent coins: A TXN output can only be referred to by a single TXN input.

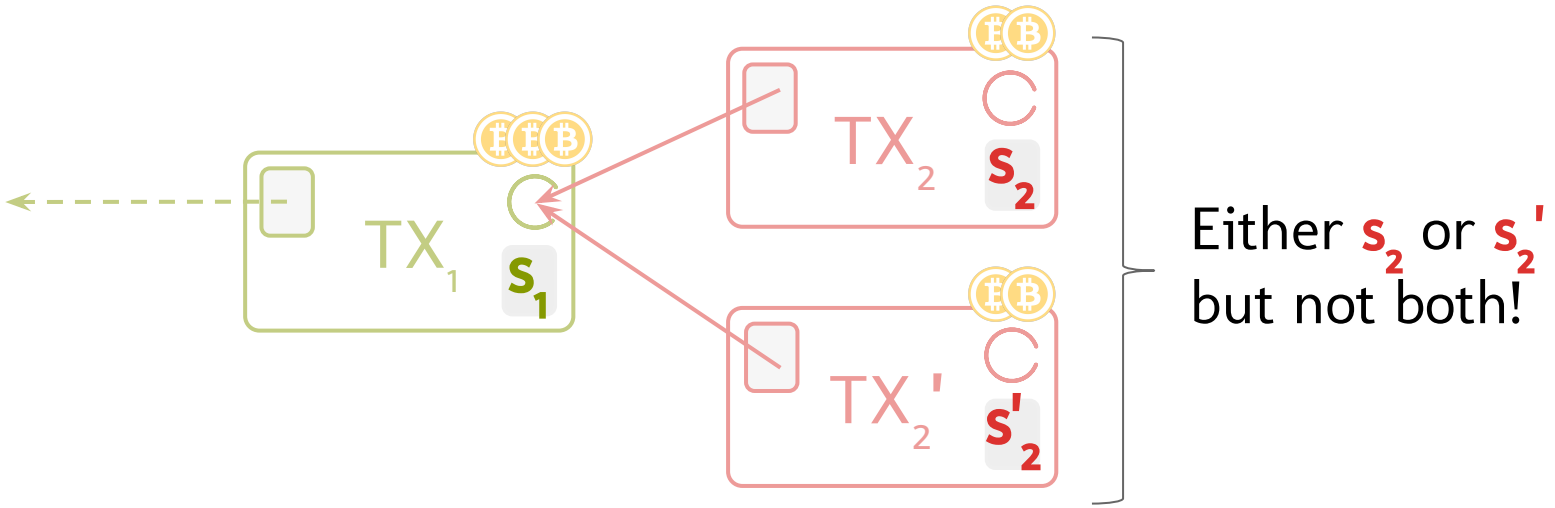
Moral of the story

Proof-of-work (PoW) consensus ⇒ No double spends



Moral of the story

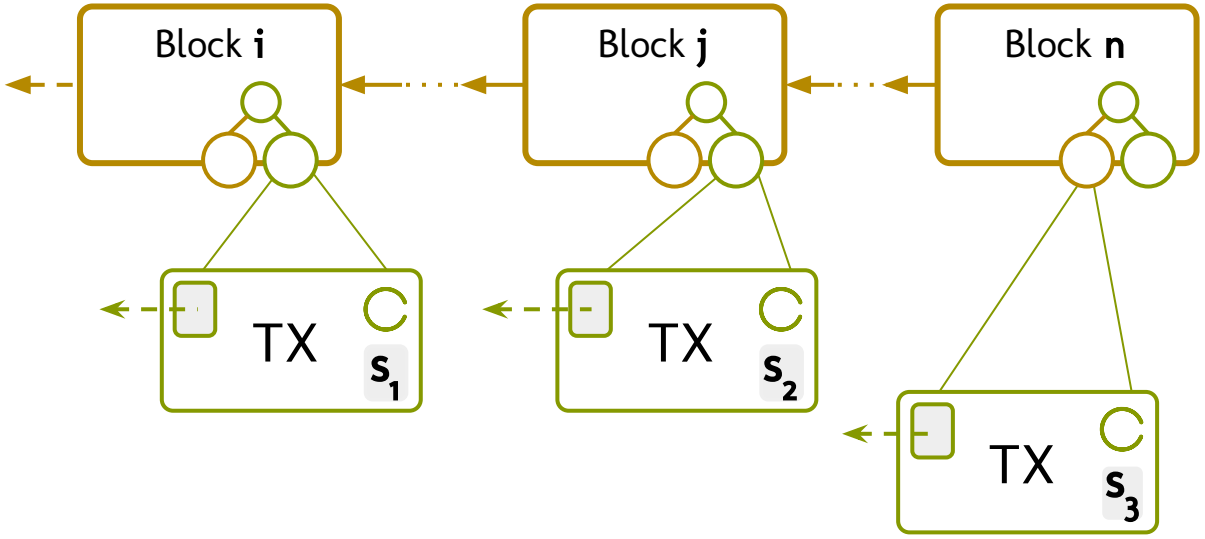
Proof-of-work (PoW) consensus ⇒ No double spends



Overview

1. What?
2. How?
   a. Bitcoin background
   b. Previous work
3. Why?

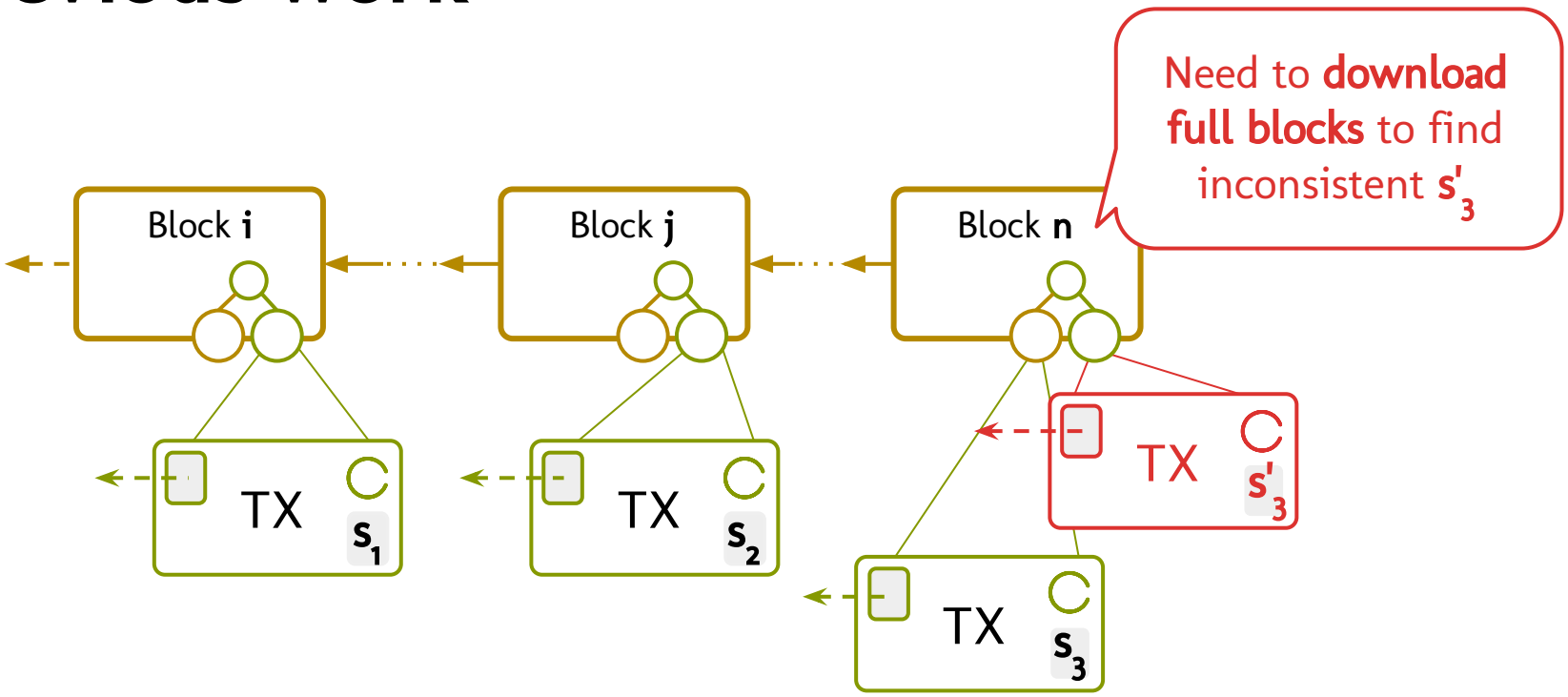
Previous work



blockstack

Keybase

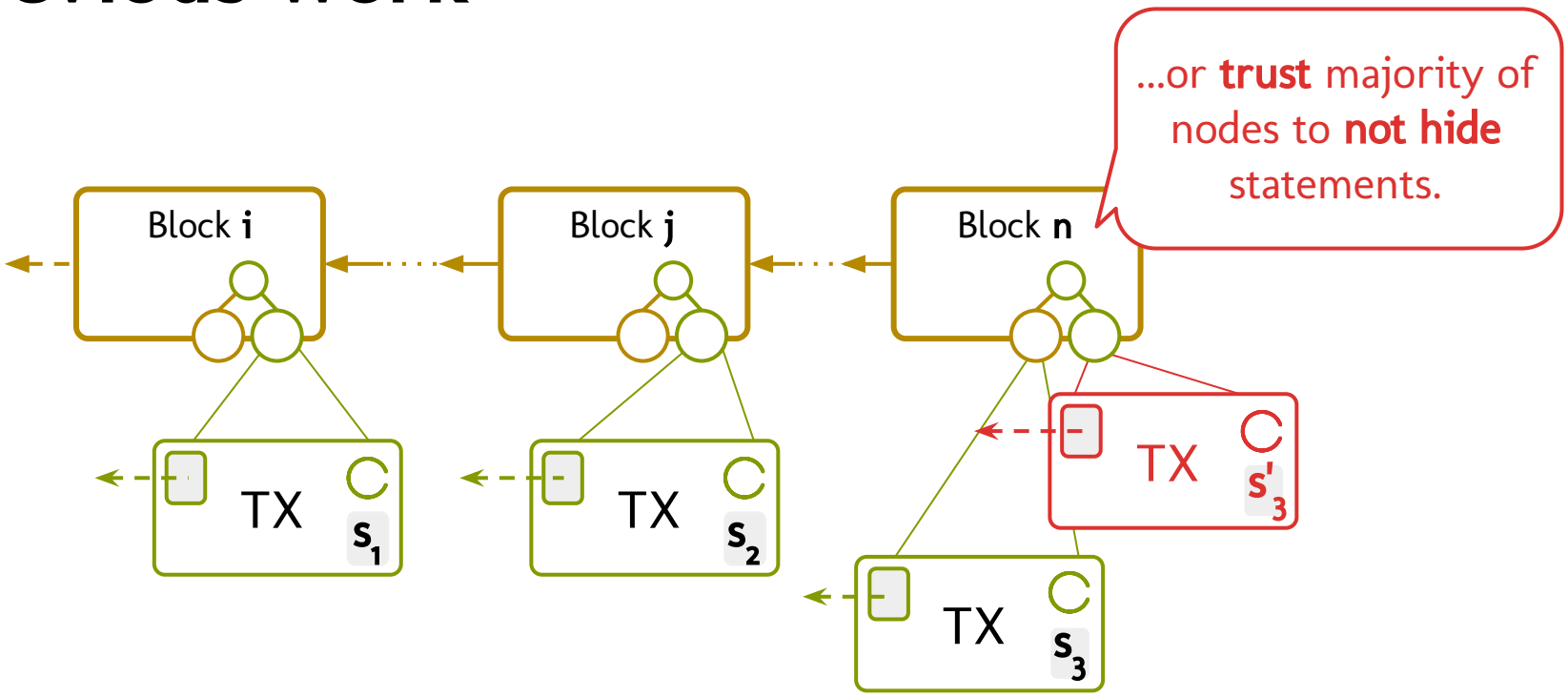
Previous work



blockstack

Keybase

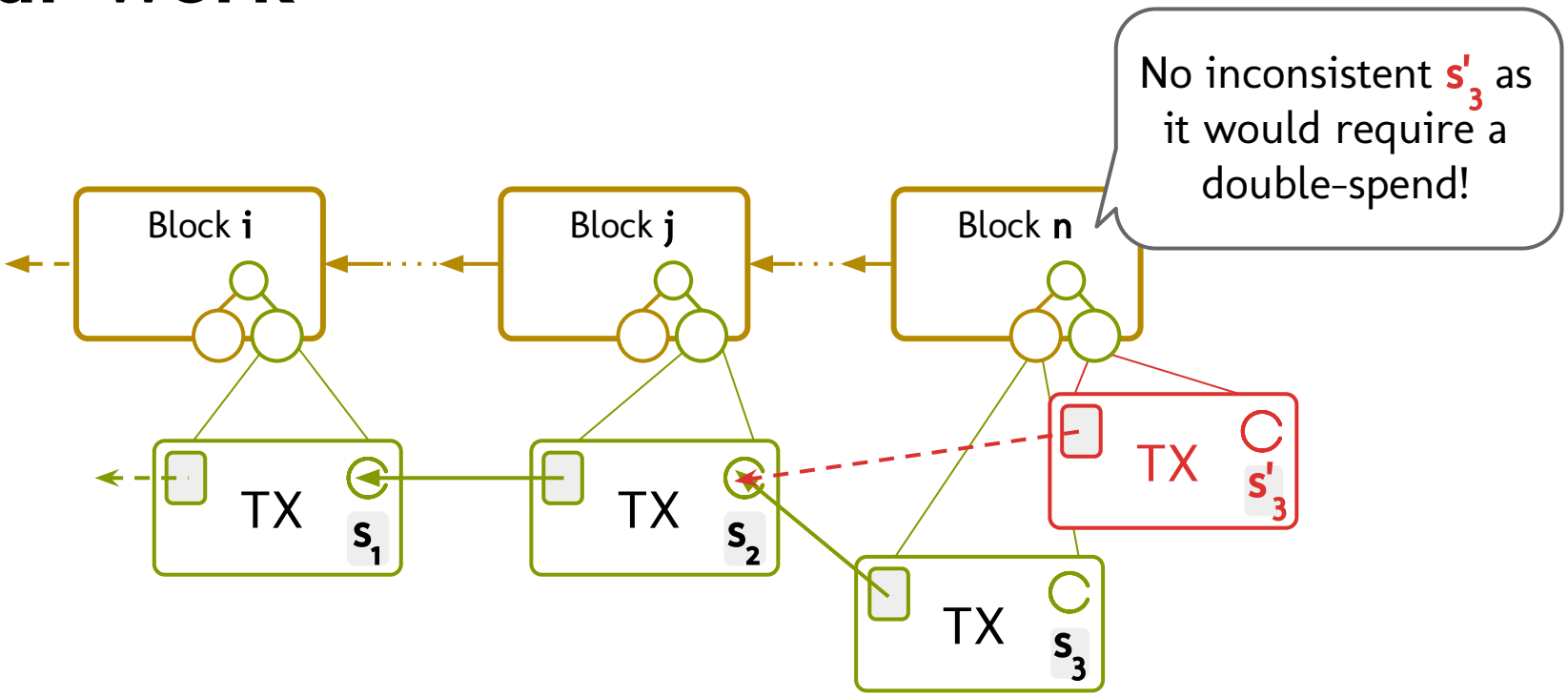
Previous work



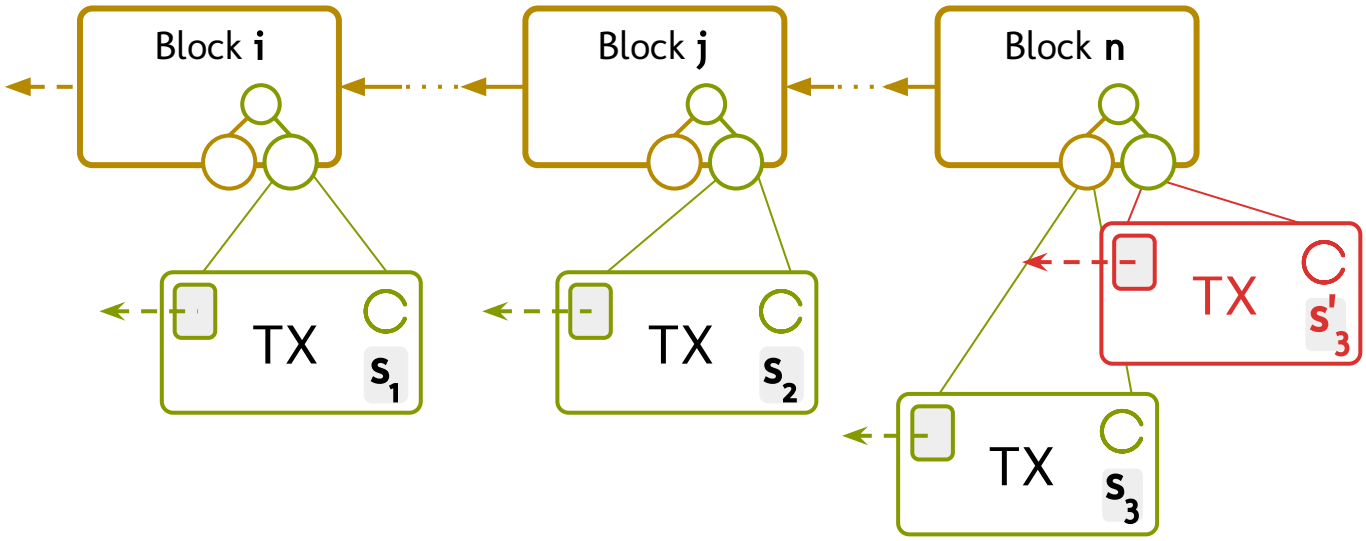
blockstack

Keybase

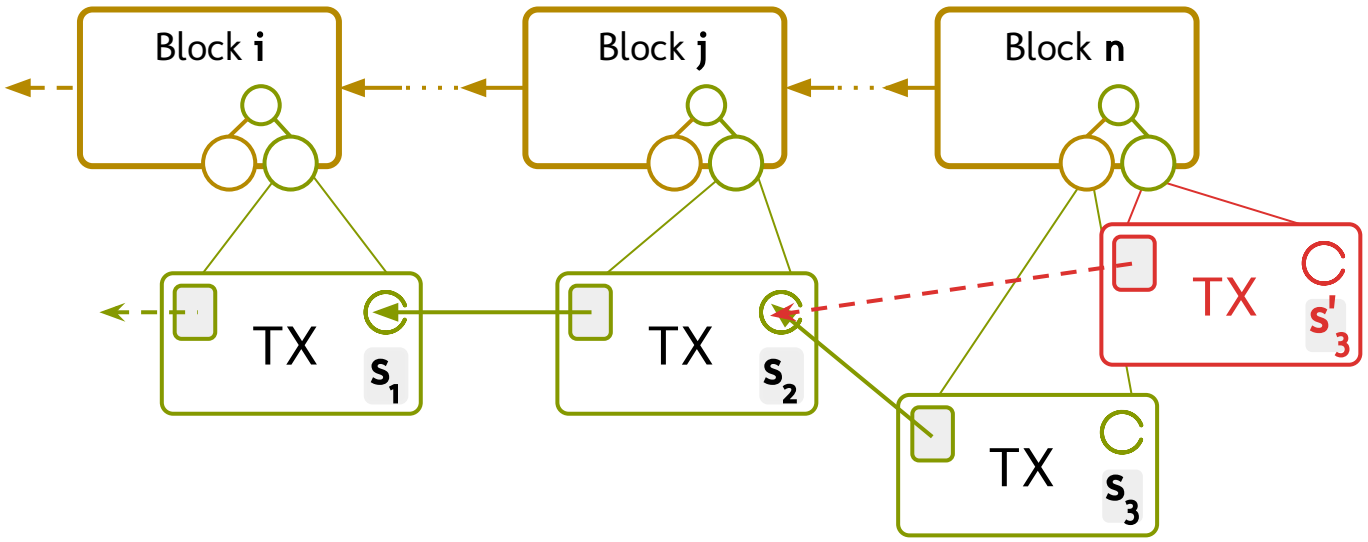
Our work


No inconsistent s'_3 as it would require a double-spend!
Block i
Block j
Block n
TX
s_1
TX
s_2
TX
s_3
TX
s'_3

Previous work



Our work



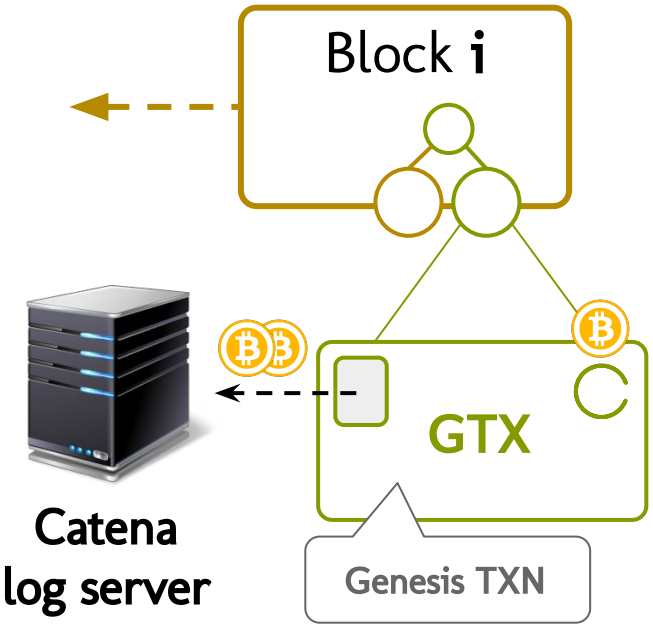
Overview

1. What?
2. How?
   a. Bitcoin background
   b. Previous work
   c. Design
3. Why?

Starting a Catena log

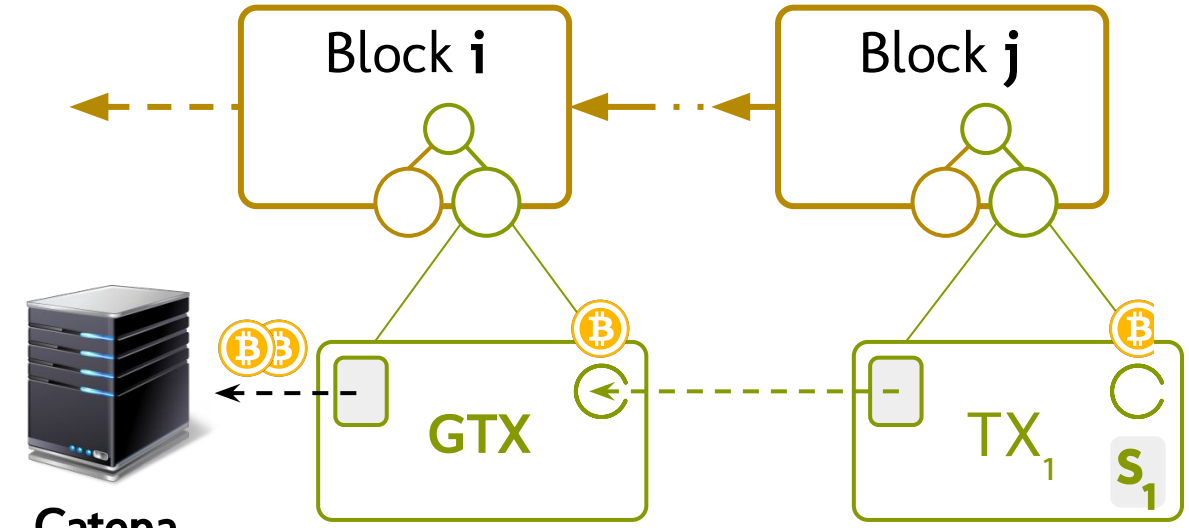
Catena
log server

Starting a Catena log



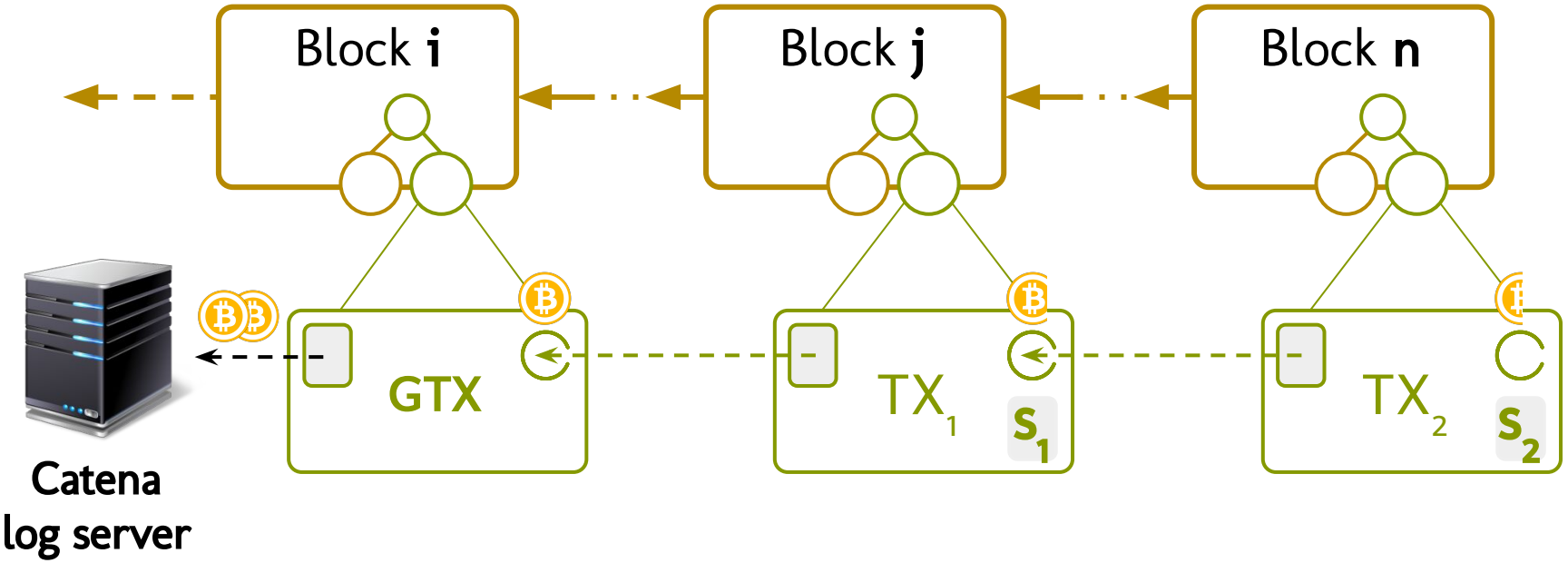
- *Genesis TXN* (GTX) = log's "public key"
- Coins from server back to server (minus fees)

Appending to a Catena log



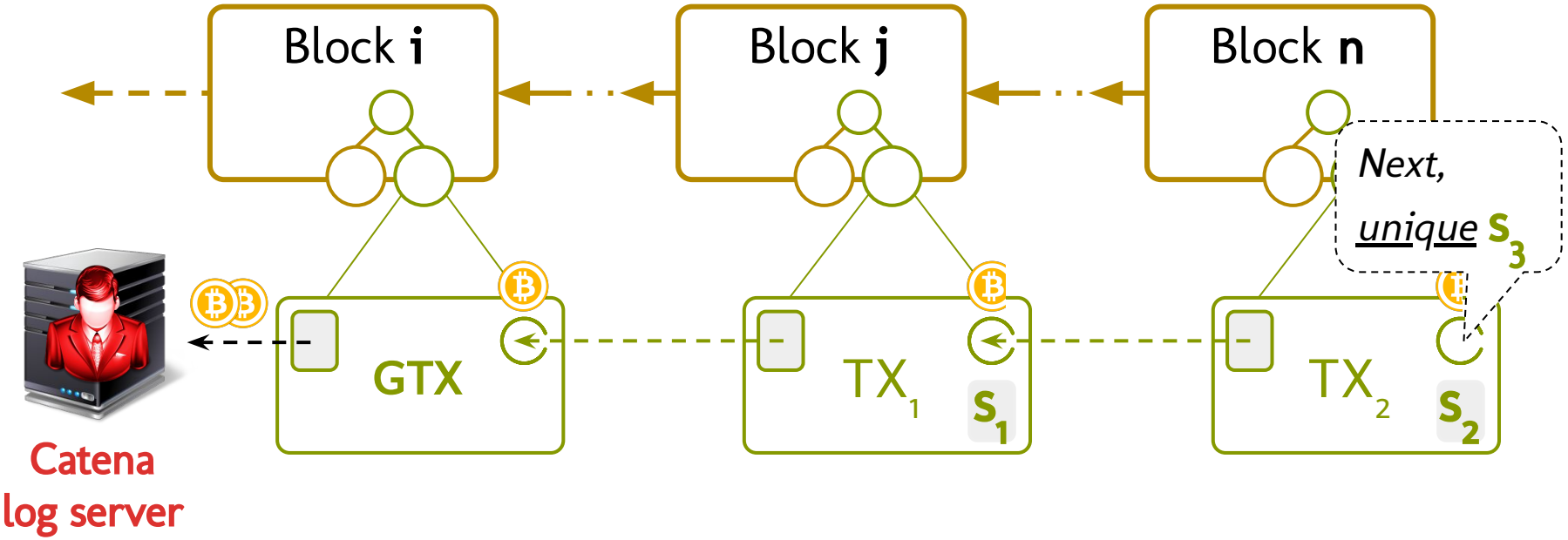
- TX_1 "spends" GTX's output, publishes s_1
- Coins from server back to server (minus fees)
- Inconsistent s_1' would require a double-spend

Appending to a Catena log



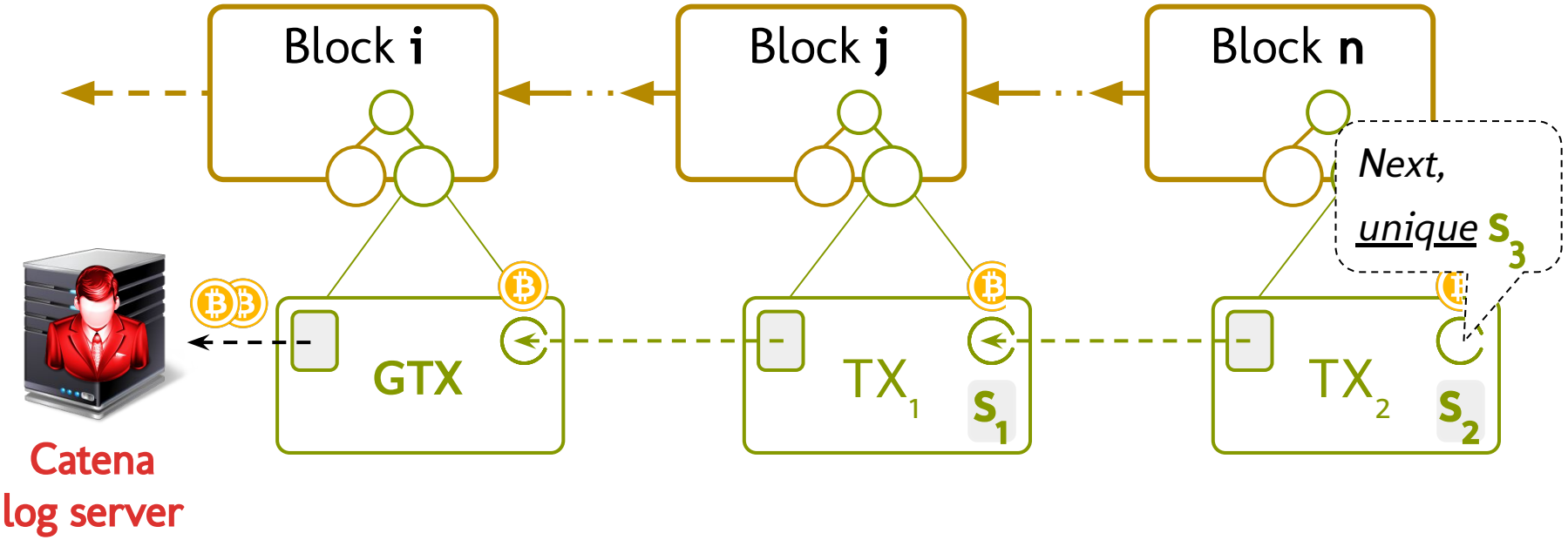
- TX_2 "spends" TX_1's output, publishes s_2
- Coins from server back to server (minus fees)
- Inconsistent s_2' would require a double-spend

Appending to a Catena log



- Server is compromised, still cannot equivocate.

Appending to a Catena log

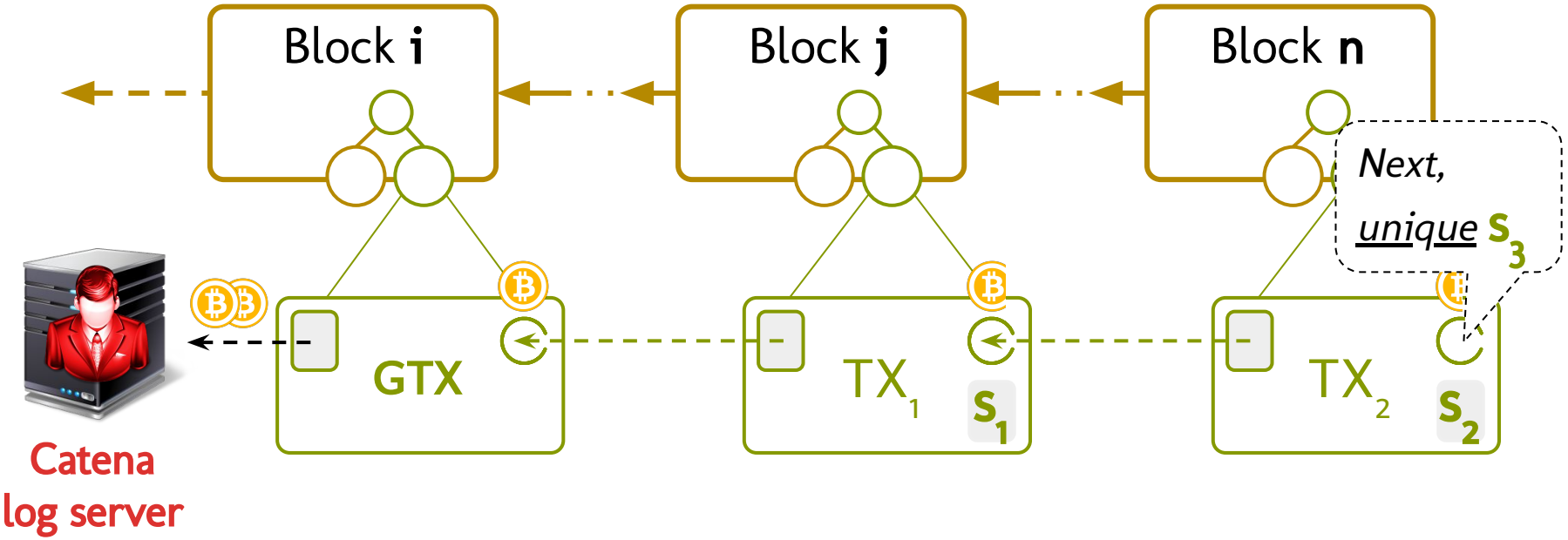


Advantages:

(1) Hard to fork

(2) Efficient to verify

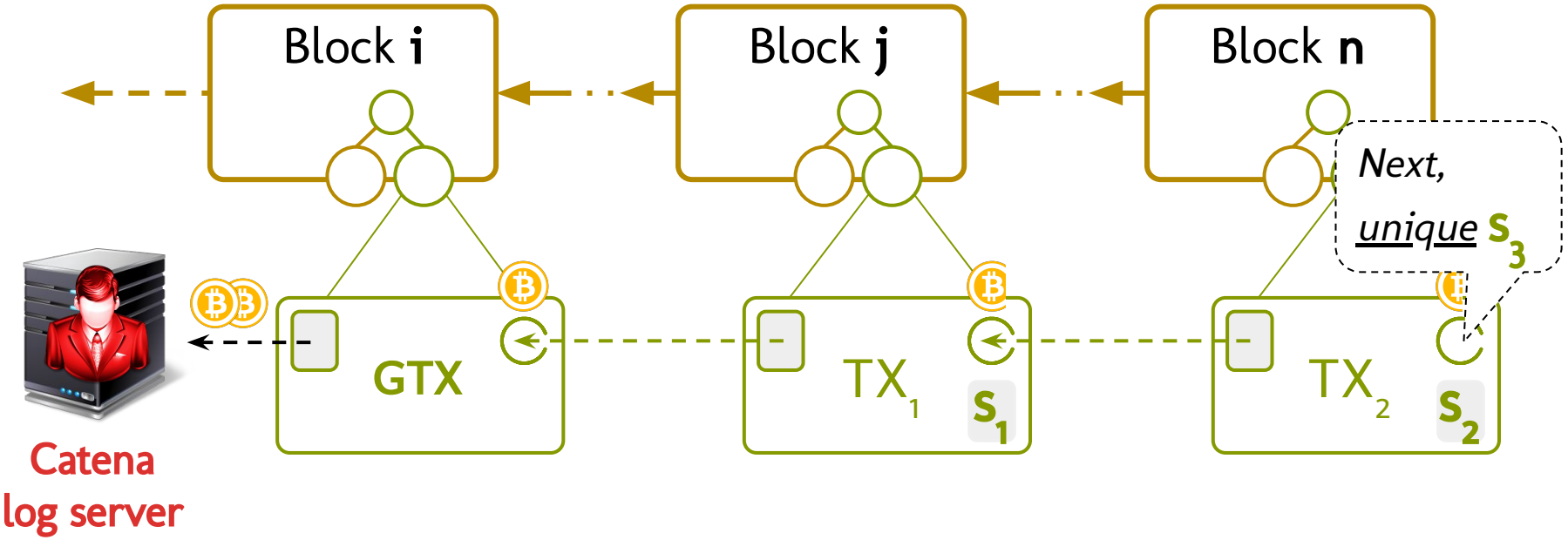
Appending to a Catena log



Advantages:
(1) Hard to fork
(2) Efficient to verify

Disadvantages:
(1) 6-block confirmation delay

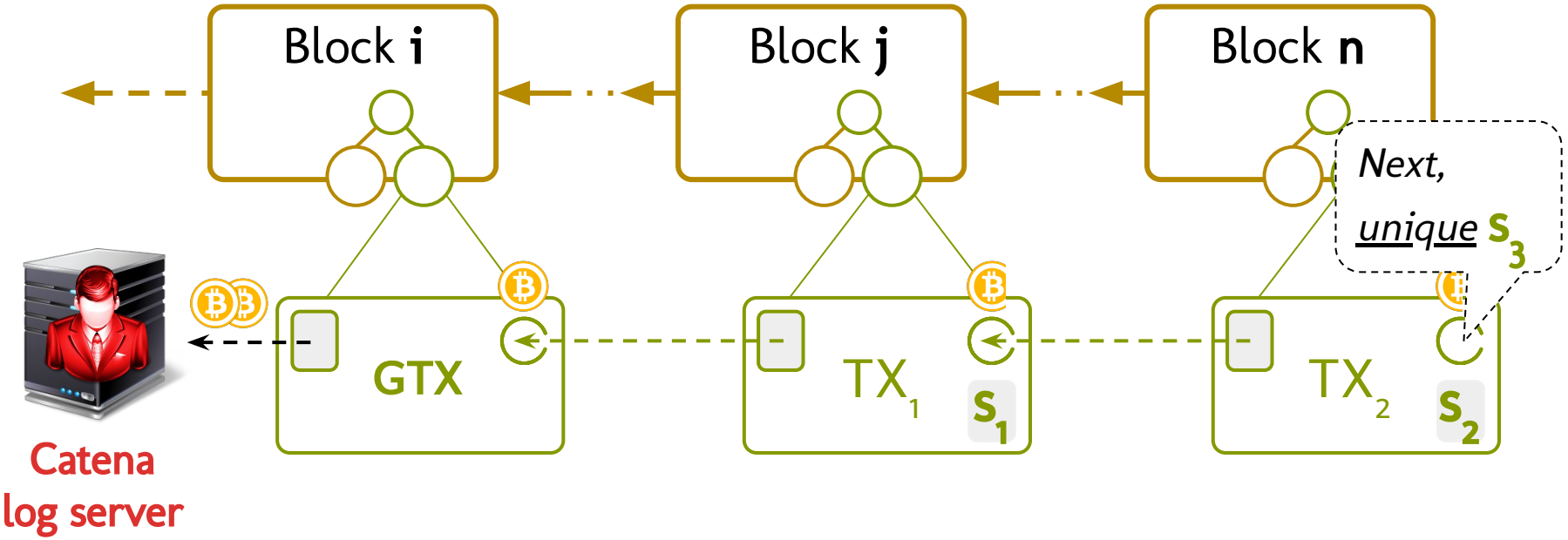
Appending to a Catena log



Advantages:
(1) Hard to fork
(2) Efficient to verify

Disadvantages:
(1) 6-block confirmation delay
(2) 1 statement every 10 minutes

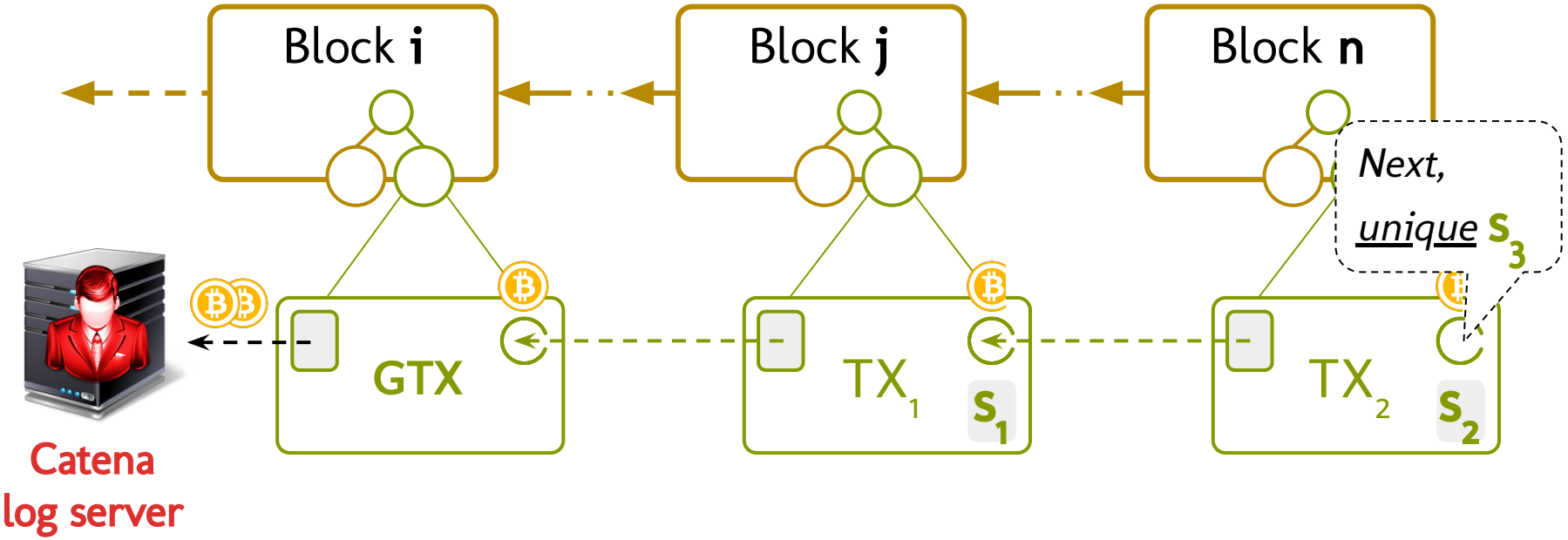
Appending to a Catena log



Advantages:
(1) Hard to fork
(2) Efficient to verify

Disadvantages:
(1) 6-block confirmation delay
(2) 1 statement every 10 minutes
(3) Must pay Bitcoin TXN fees

Appending to a Catena log



Advantages:
(1) Hard to fork
(2) Efficient to verify

Disadvantages:
(1) 6-block confirmation delay
(2) 1 statement every 10 minutes
(3) Must pay Bitcoin TXN fees
(4) No freshness guarantee

Overview

1. Catena: What?
2. Catena: How?
   a. Bitcoin background
   b. Previous work
   c. Design
   d. **Efficient auditing**
3. Potentially-interesting applications

Efficient auditing

Efficient auditing

Catena
client

Catena
log server

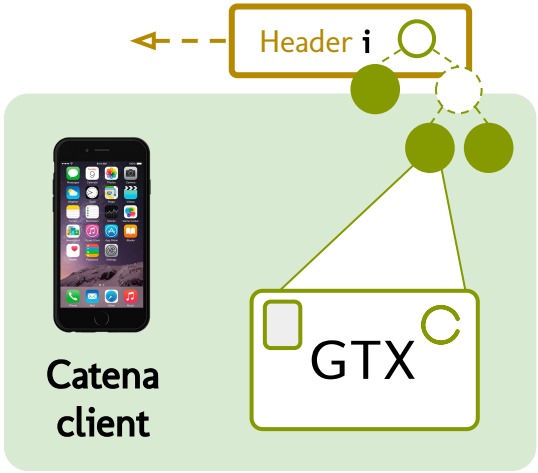
Efficient auditing

Catena
client

Bitcoin P2P
(11,500 nodes)

Catena
log server

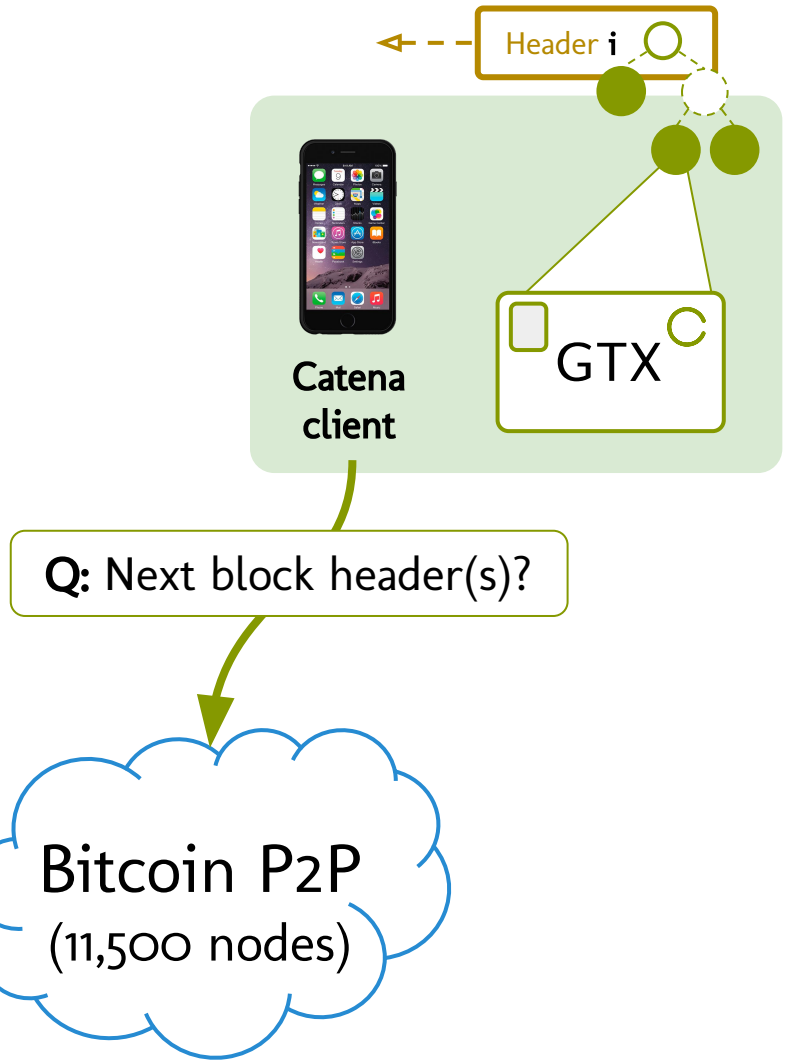
Efficient auditing



Bitcoin P2P
(11,500 nodes)

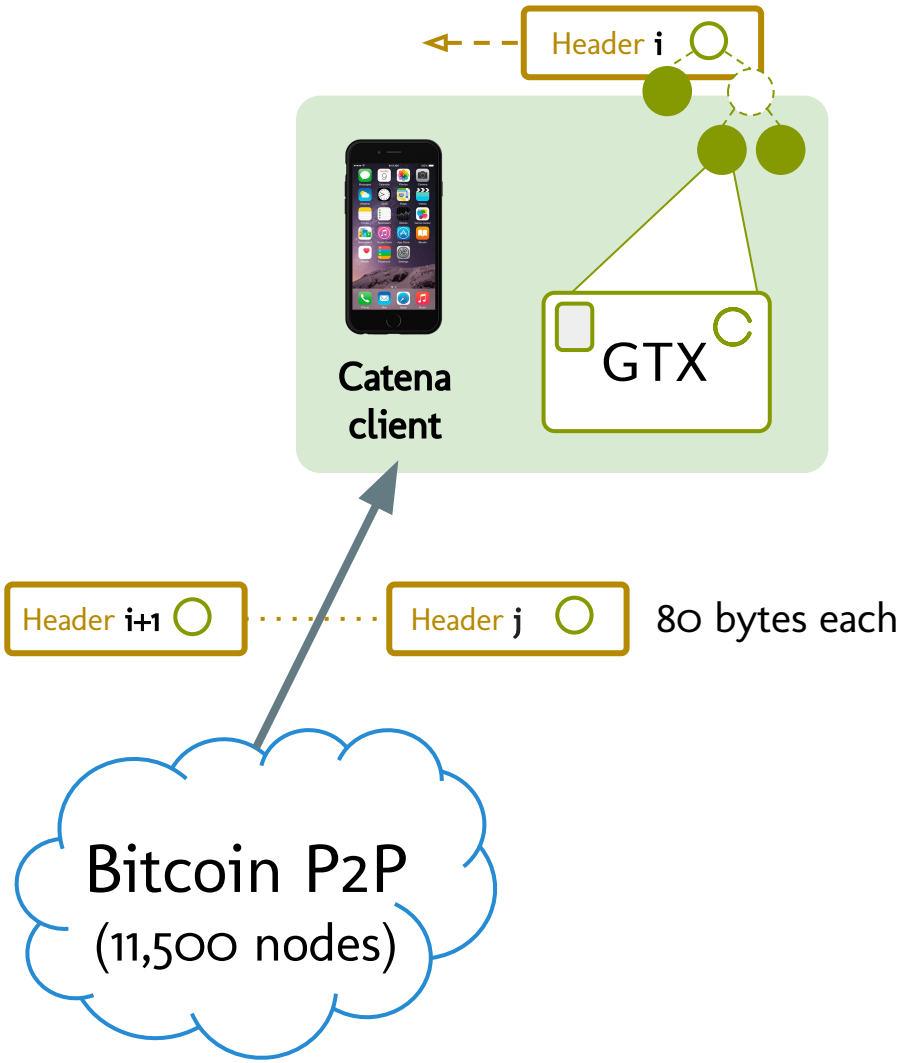
Catena
log server

Efficient auditing



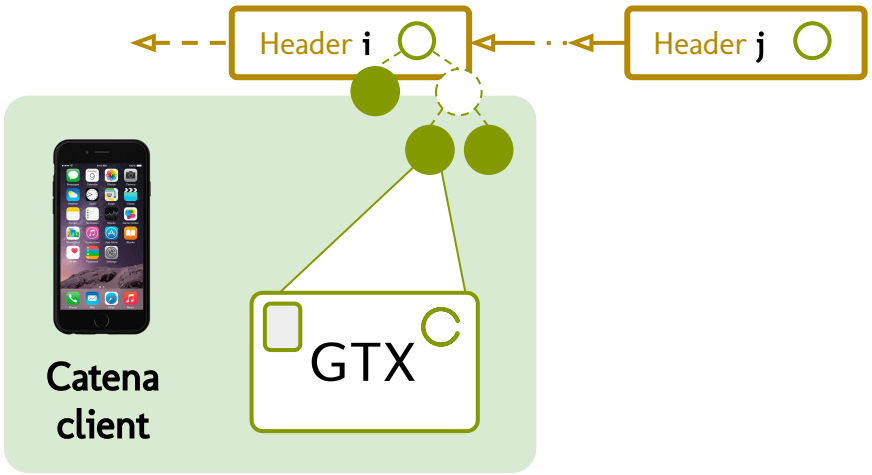
Catena log server

Efficient auditing



Catena log server

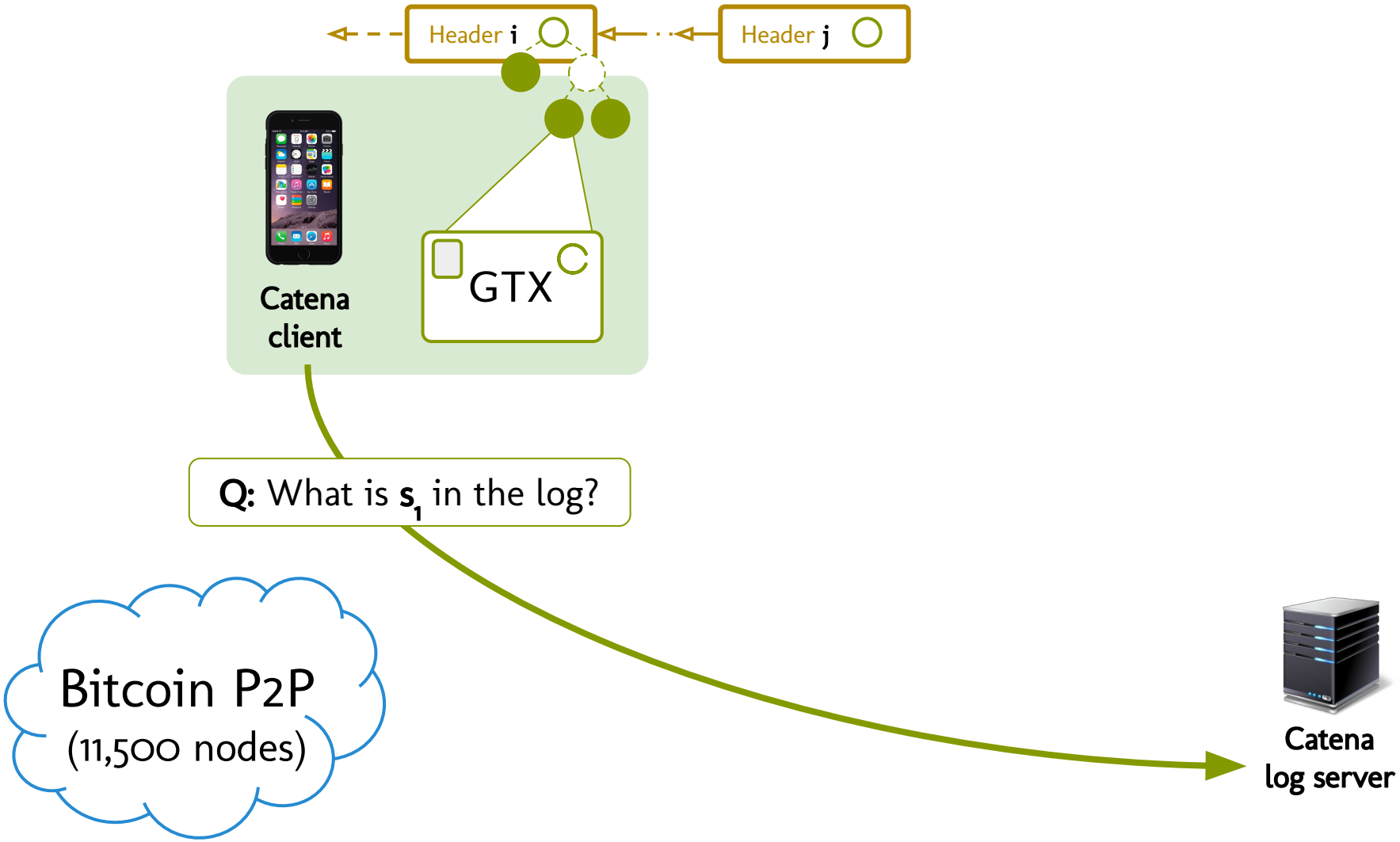
Efficient auditing



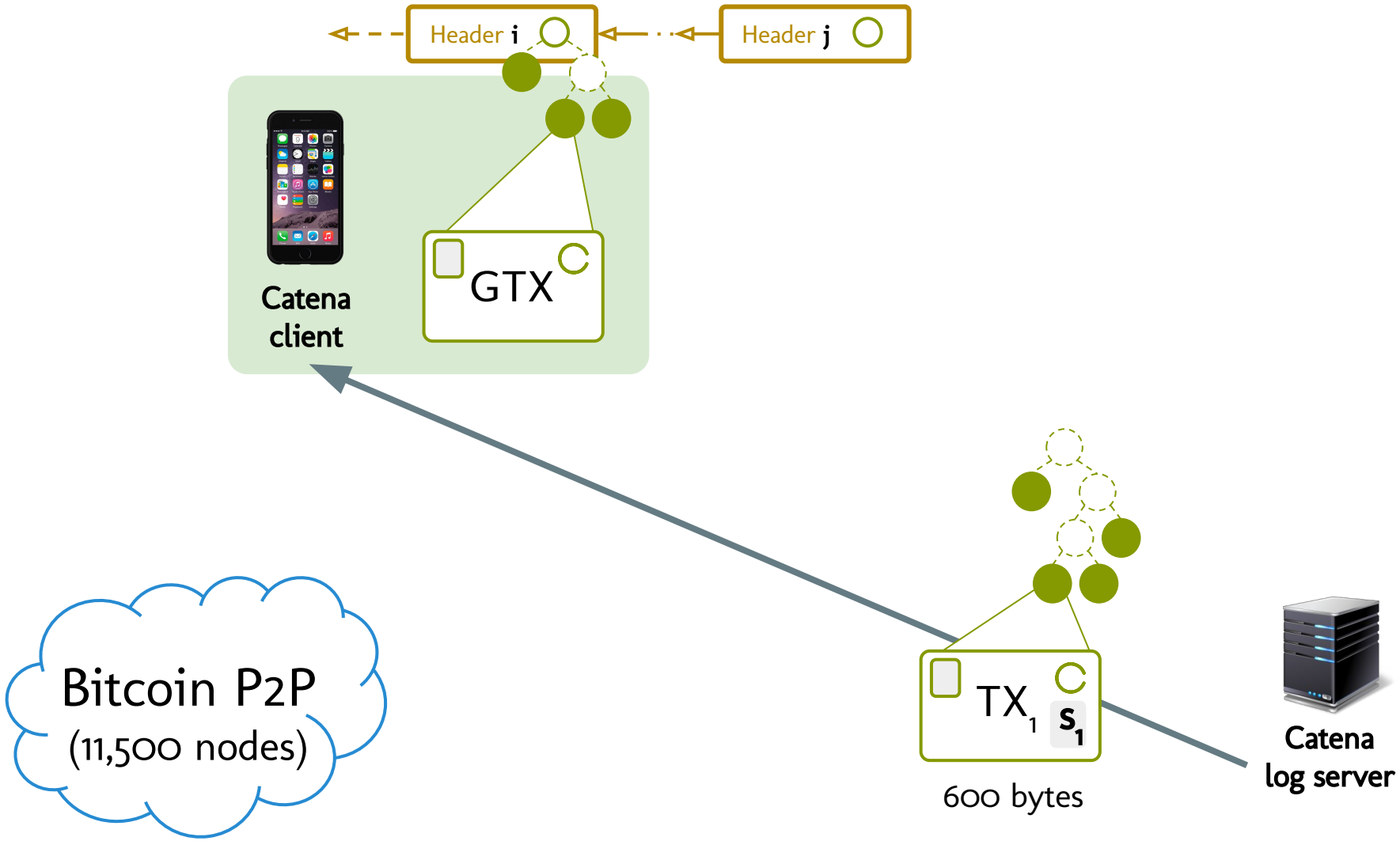
Bitcoin P2P
(11,500 nodes)

Catena log server

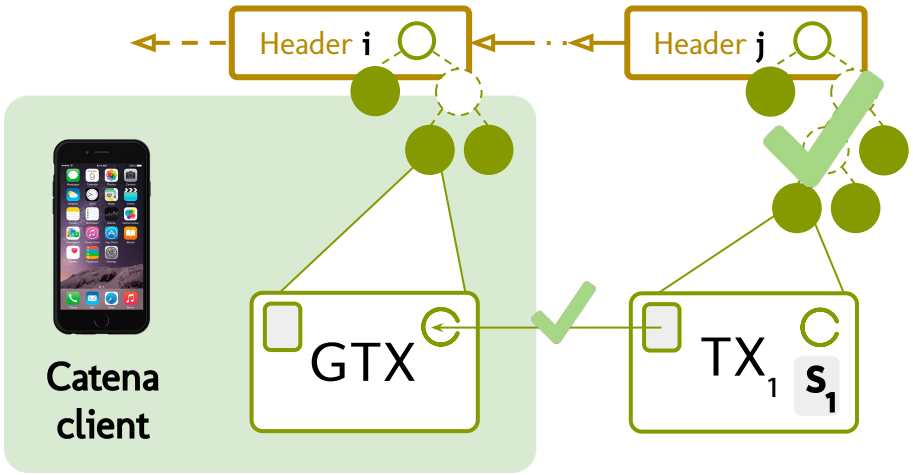
Efficient auditing



Efficient auditing



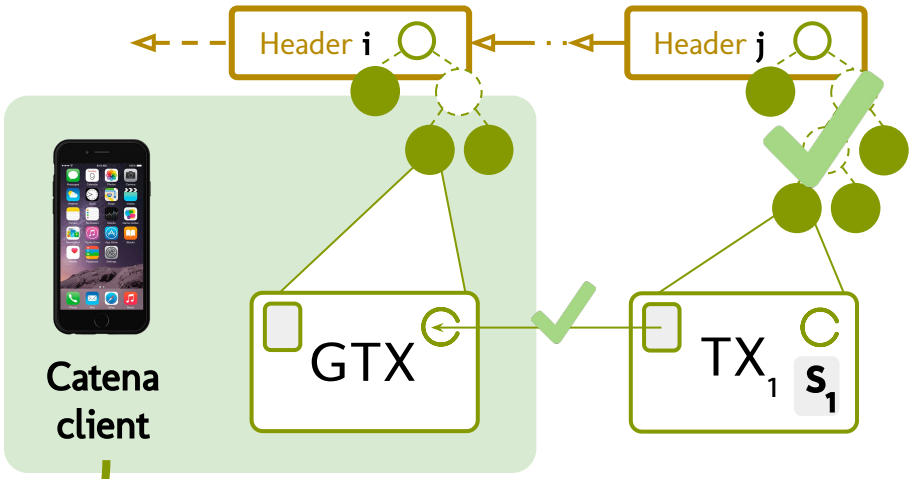
Efficient auditing



Bitcoin P2P
(11,500 nodes)

Catena log server

Efficient auditing

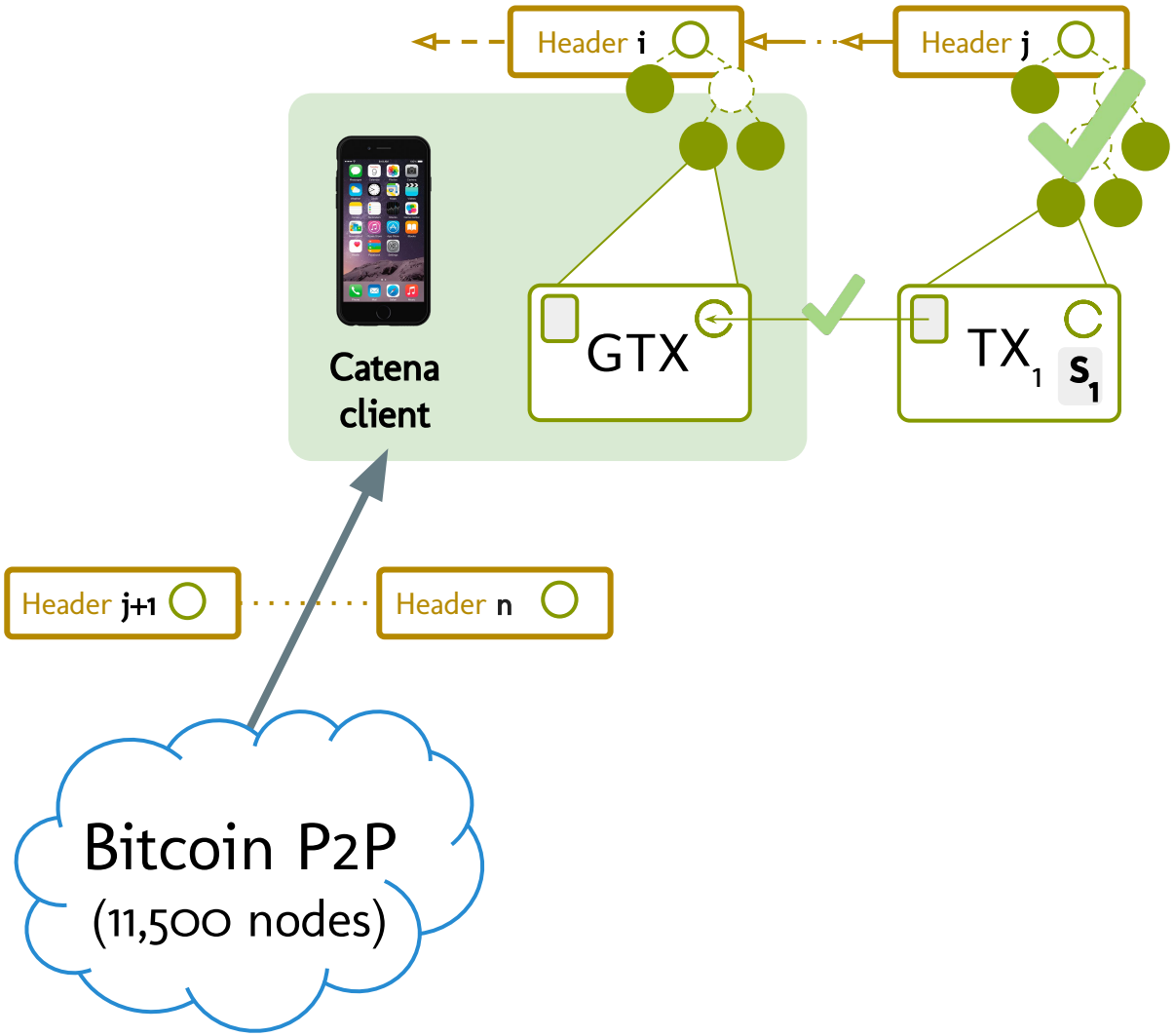


Q: Next block header(s)?

Bitcoin P2P
(11,500 nodes)

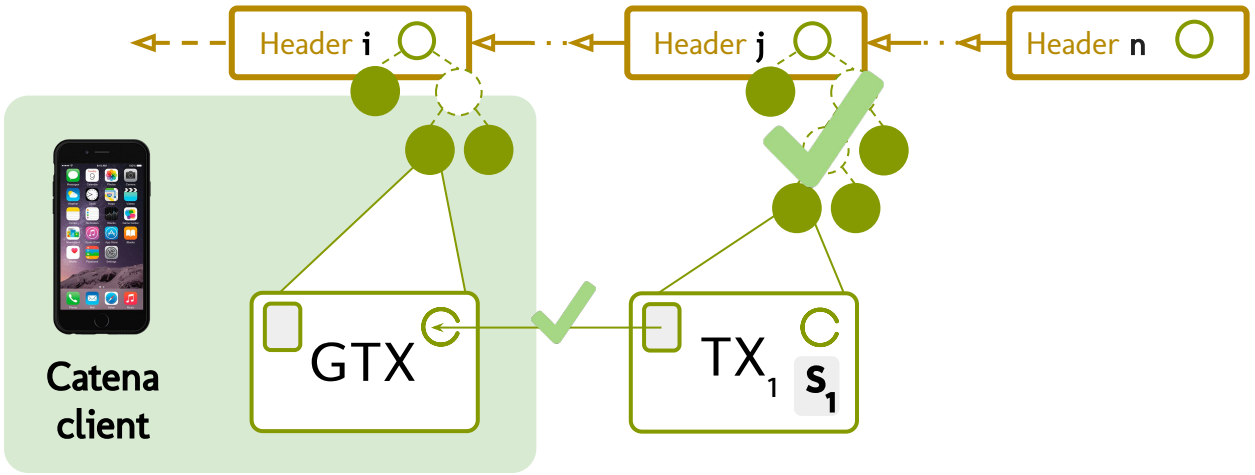
Catena log server

Efficient auditing



Catena log server

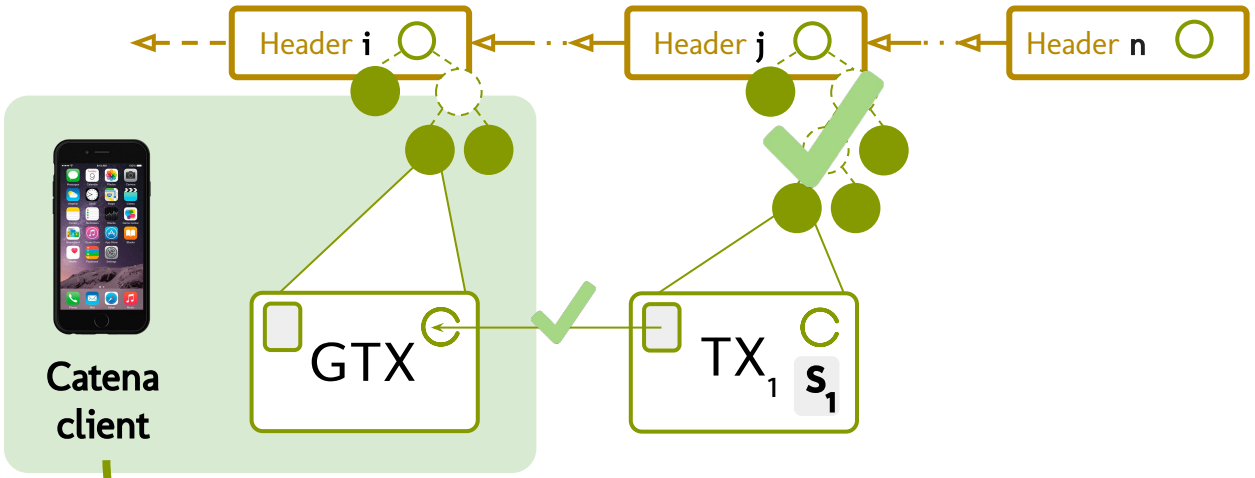
Efficient auditing



Bitcoin P2P
(11,500 nodes)

Catena log server

Efficient auditing

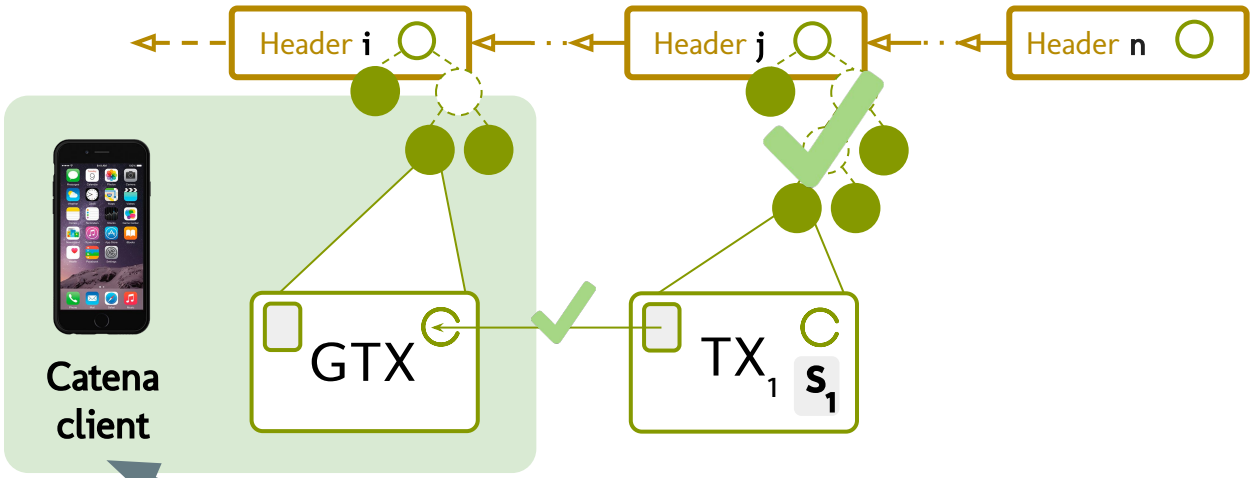


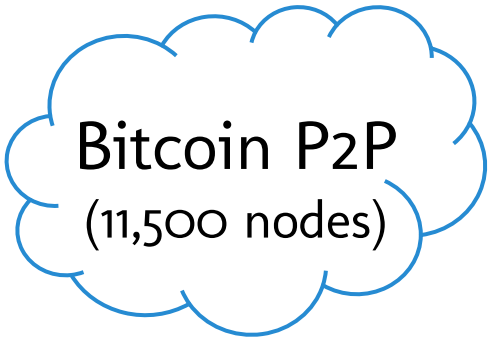
Q: What is s_2 in the log?

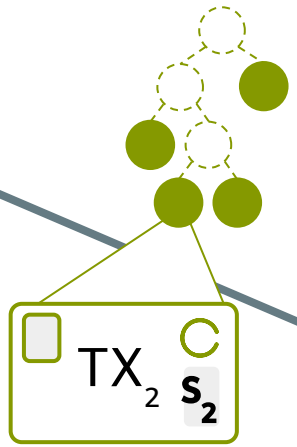
Bitcoin P2P
(11,500 nodes)

Catena log server

Efficient auditing

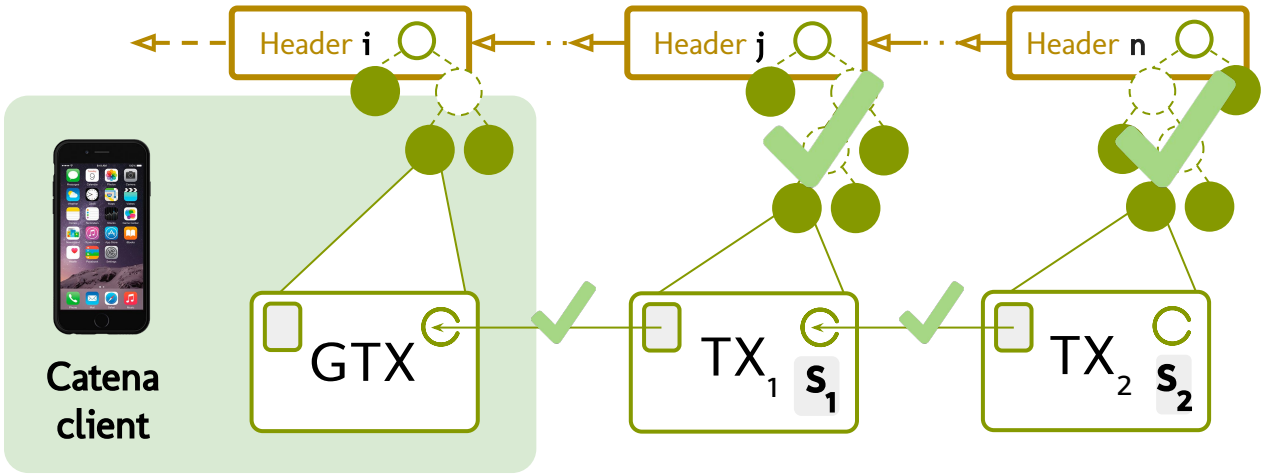






Catena log server

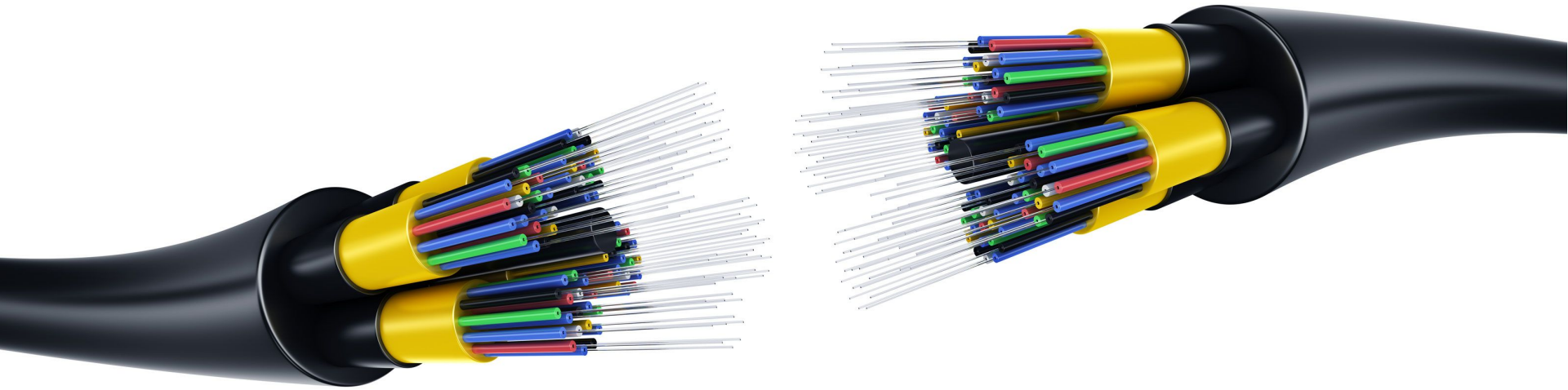
Efficient auditing



Bitcoin P2P
(11,500 nodes)

Catena log server

Auditing bandwidth



e.g., **500K** block headers + **10K** statements = **~46** MB

(80 bytes each) (around 600 bytes each)

Overview

1. What?
2. How?
   a. Bitcoin background
   b. Previous work
   c. Design
   d. Efficient auditing
   e. Scalability
3. Why?

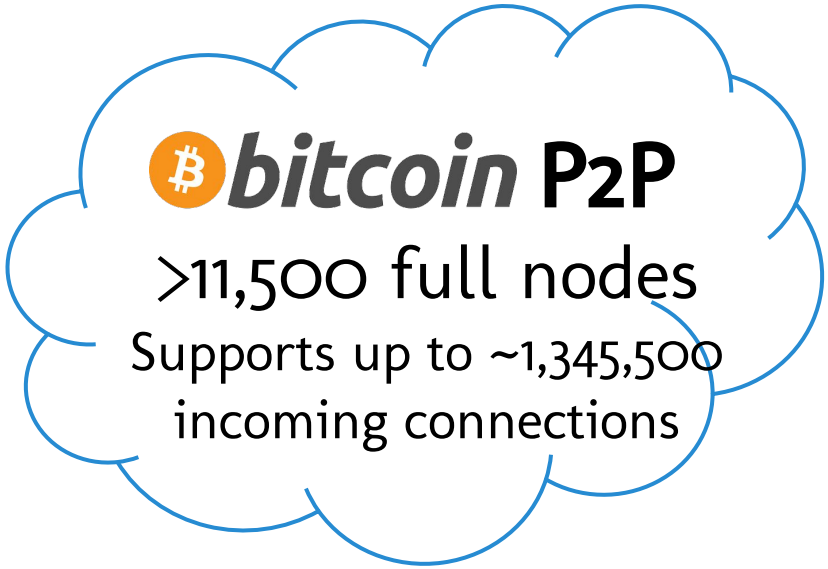
Catena scalability

Catena client 1

Catena client 2

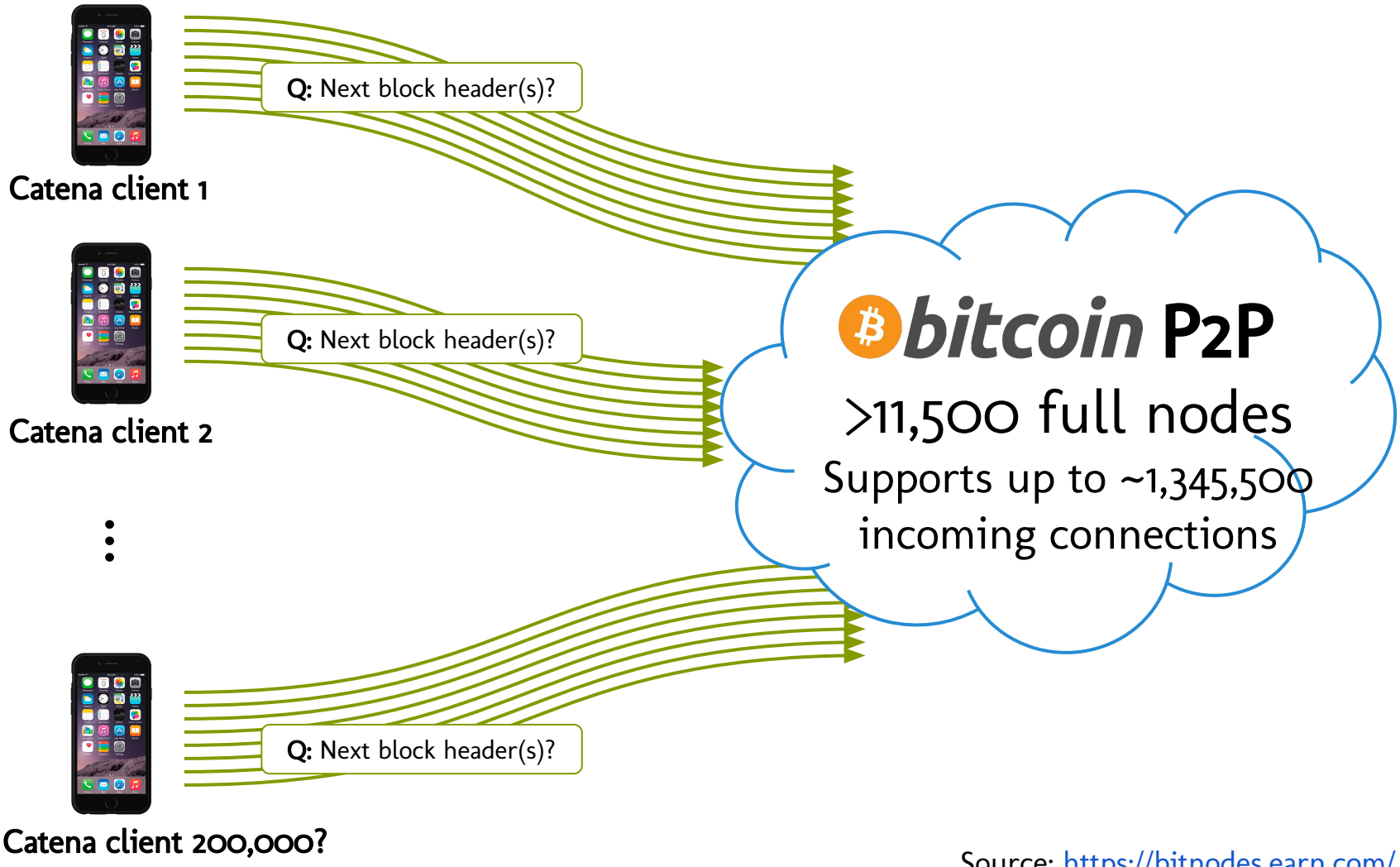
⋮

Catena client 200,000?



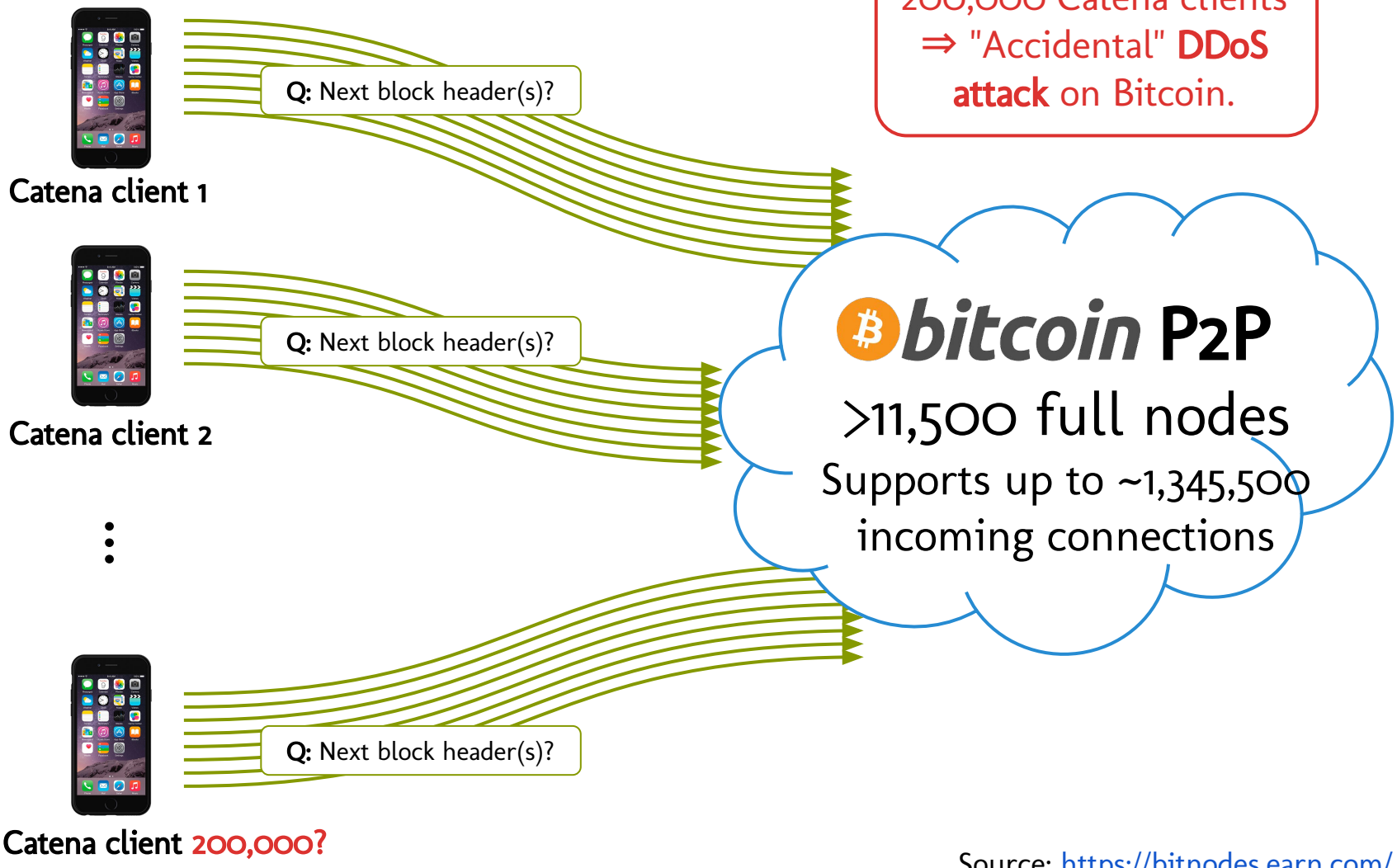
Source: https://bitnodes.earn.com/

Catena scalability



Source: https://bitnodes.earn.com/

Catena scalability



Source: https://bitnodes.earn.com/

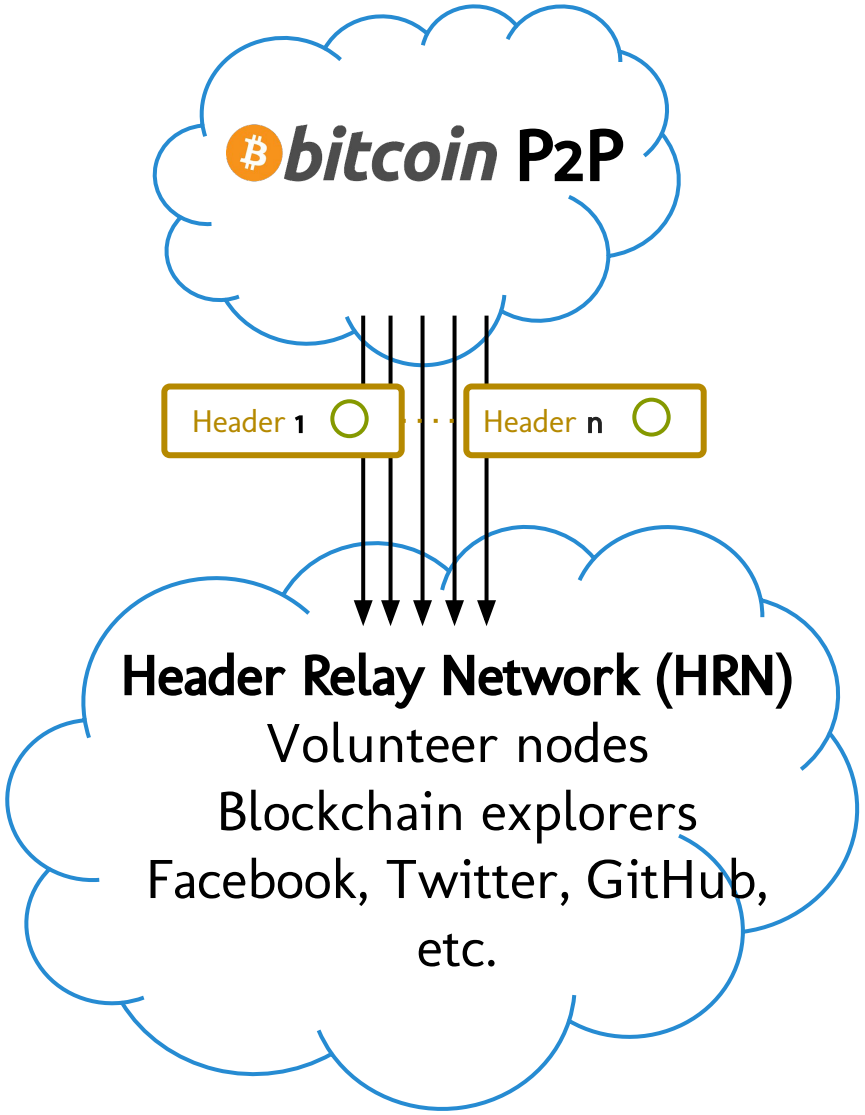
Catena scalability

Catena client 1

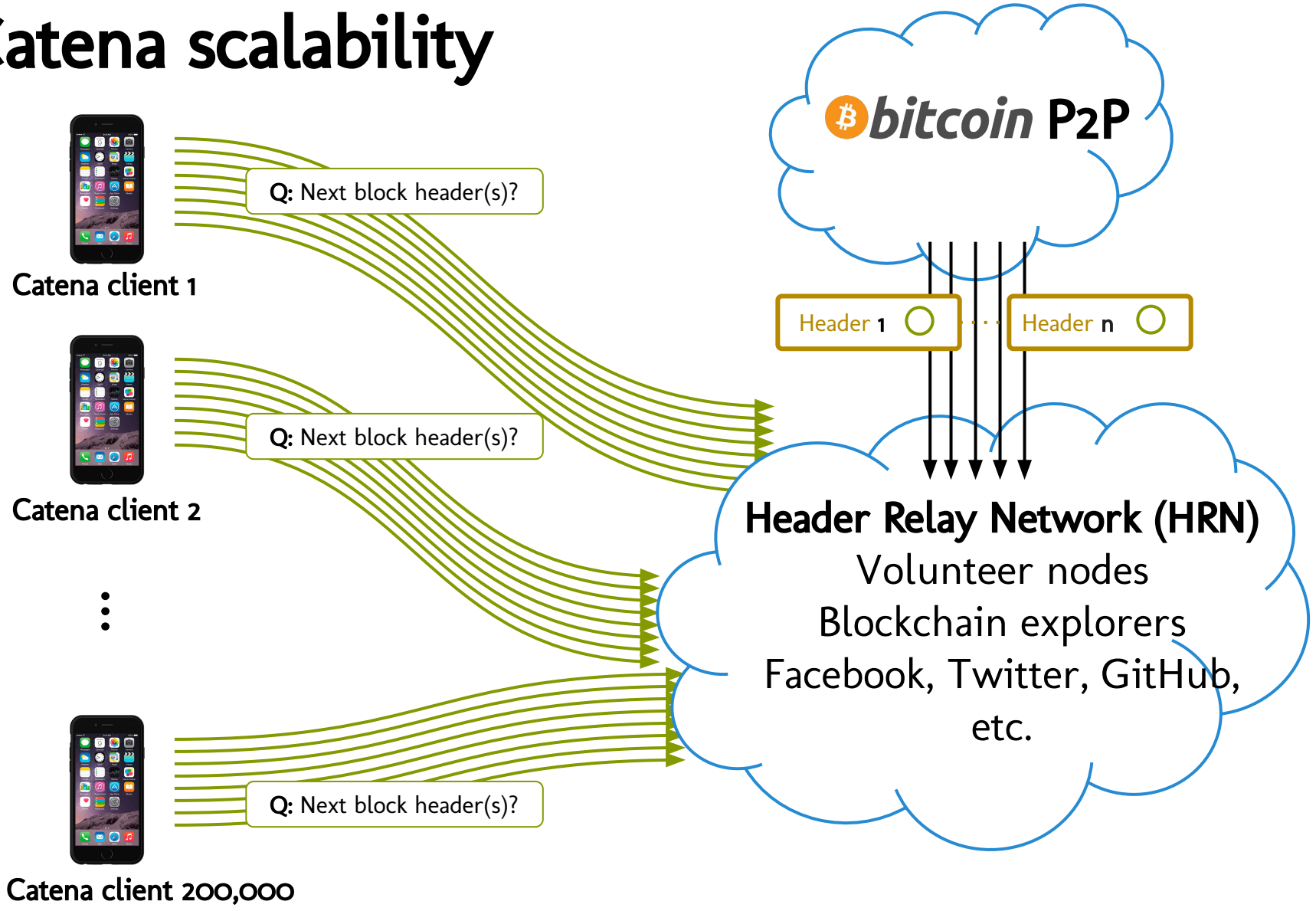
Catena client 2

⋮

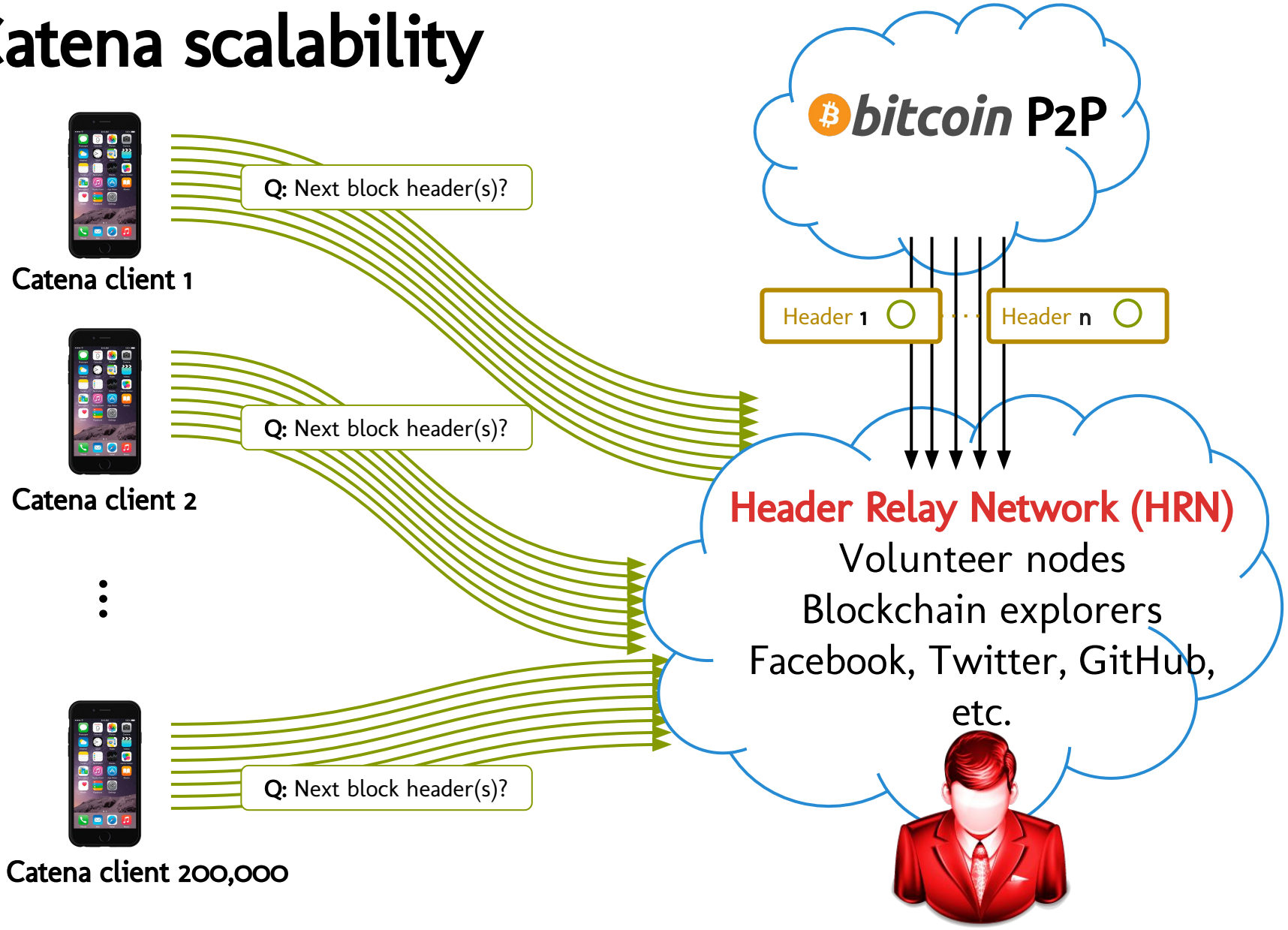
Catena client 200,000



Catena scalability



Catena scalability



The cost of a statement

The cost of a statement

To append a statement, must issue TXN and pay fee.

The cost of a statement

To append a statement, must issue TXN and pay fee.

TXN size: 235-byte

The cost of a statement

To append a statement, must issue TXN and pay fee.

TXN size: 235-byte

Fee as of Dec 13th, 2017**:** $16.24 (10 mins)

Latest fees: https://bitcoinfees.info/

The cost of a statement

To append a statement, must issue TXN and pay fee.

TXN size: 235-byte

Fee as of Dec 13th, 2017**:** $16.24 (10 mins)

Fee as of Feb 28th, 2017**:** $0.78 (10 mins)

Latest fees: https://bitcoinfees.info/

The cost of a statement

To append a statement, must issue TXN and pay fee.

TXN size: 235-byte

Fee as of Dec 13th, 2017**:** $16.24 (10 mins)

Fee as of Feb 28th, 2017**:** $0.78 (10 mins)

PS: Statements can be "batched" using Merkle trees.

Latest fees: https://bitcoinfees.info/

Overview

1. What?
2. How?
3. Why?
    a. Secure software update

Secure software update

Secure software update

Example attack:

(1) Compromise bitcoin.org (or the network)

Secure software update

Example attack:

(1) Compromise bitcoin.org (or the network)

(2) Change the Bitcoin binary to your malicious binary

Secure software update

Example attack:

(1) Compromise bitcoin.org (or the network)

(2) Change the Bitcoin binary to your malicious binary

(3) Wait for people to install your malicious Bitcoin binary

Secure software update

Example attack:

(1) Compromise bitcoin.org (or the network)

(2) Change the Bitcoin binary to your malicious binary

(3) Wait for people to install your malicious Bitcoin binary

(4) Steal their coins, steal their data, etc.

Secure software update

Example attack:

(1) Compromise bitcoin.org (or the network)

(2) Change the Bitcoin binary to your malicious binary

(3) Wait for people to install your malicious Bitcoin binary

(4) Steal their coins, steal their data, etc.

Example: bitcoin.org, "0.13.0 Binary Safety Warning," August 17th, 2016

Secure software update

Example attack:

(1) Compromise bitcoin.org (or the network)

(2) Change the Bitcoin binary to your malicious binary

(3) Wait for people to install your malicious Bitcoin binary

(4) Steal their coins, steal their data, etc.

Example: bitcoin.org, "0.13.0 Binary Safety Warning," August 17th, 2016

Typical defense: Devs sign Bitcoin binaries with SK and protect SK.

Secure software update

Example attack:

(1) Compromise bitcoin.org (or the network)

(2) Change the Bitcoin binary to your malicious binary

(3) Wait for people to install your malicious Bitcoin binary

(4) Steal their coins, steal their data, etc.

Example: bitcoin.org, "0.13.0 Binary Safety Warning," August 17th, 2016

Typical defense: Devs sign Bitcoin binaries with SK and protect SK.

Problem: (1) Not everyone checks sig. (2) Hard to detect stolen SK.

Secure software update

Example attack:

(1) Compromise bitcoin.org (or the network)

(2) Change the Bitcoin binary to your malicious binary

(3) Wait for people to install your malicious Bitcoin binary

(4) Steal their coins, steal their data, etc.

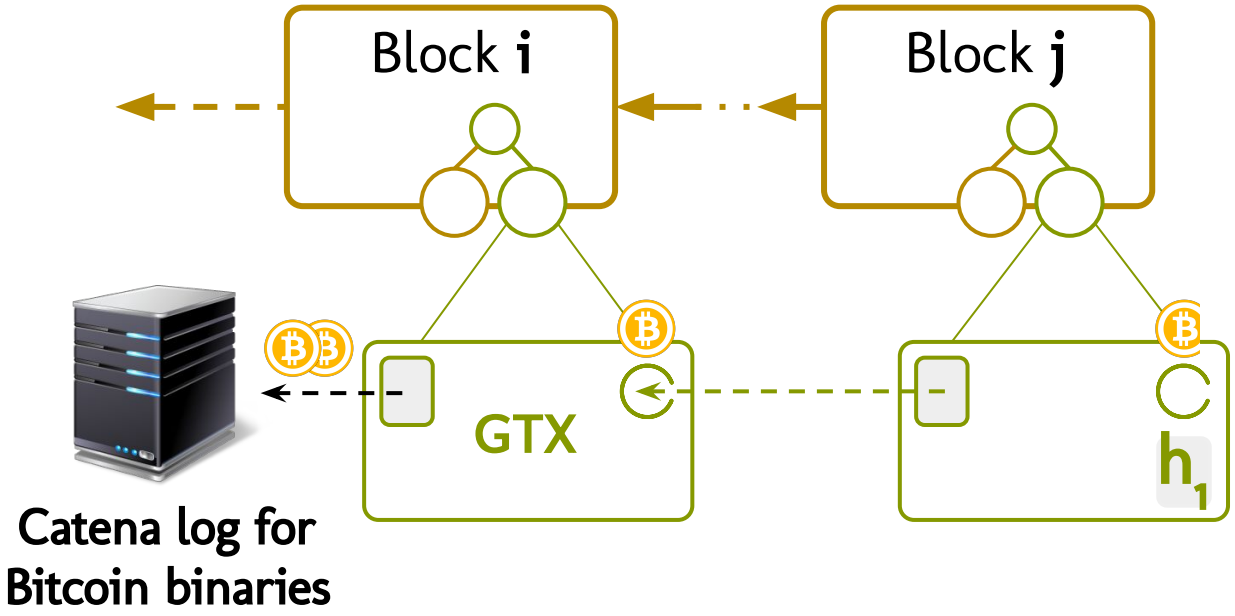
Example: bitcoin.org, "0.13.0 Binary Safety Warning," August 17th, 2016

Typical defense: Devs sign Bitcoin binaries with SK and protect SK.

Problem: (1) Not everyone checks sig. (2) Hard to detect stolen SK.

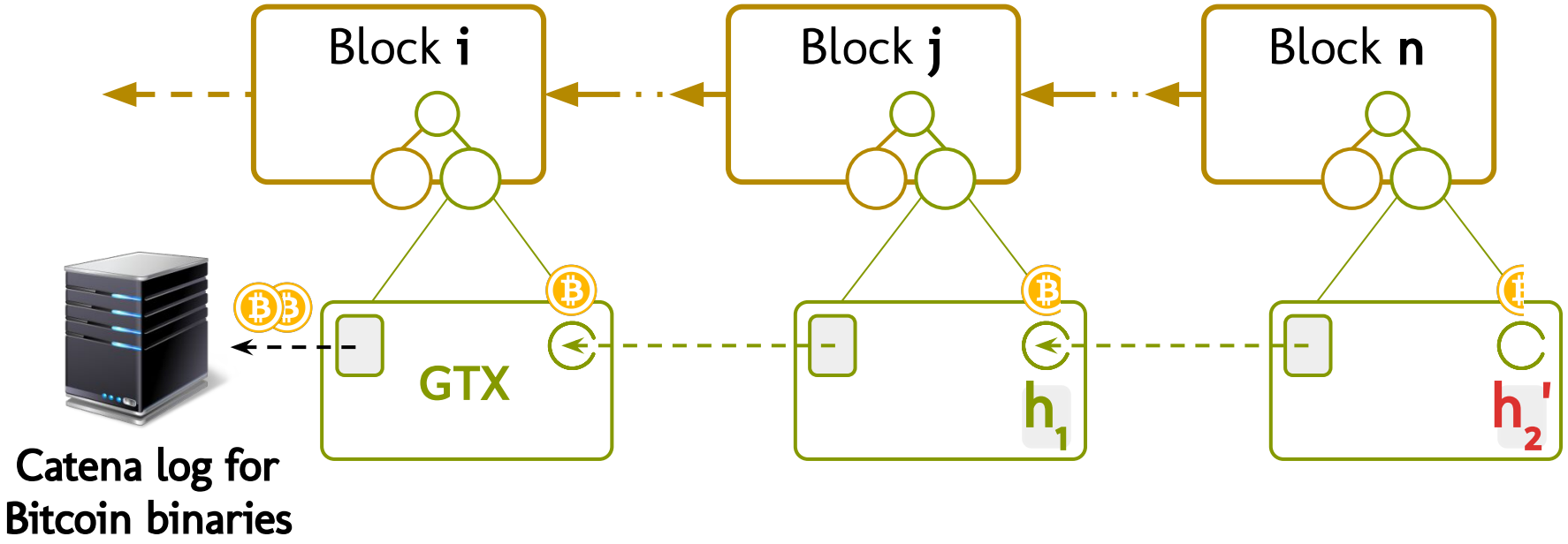
Solution: Publish signatures in a Catena log ⇒ **Can at least detect.**

Software transparency to the rescue



h_1 = SHA256(bitcoin-0.0.1.tar.gz)

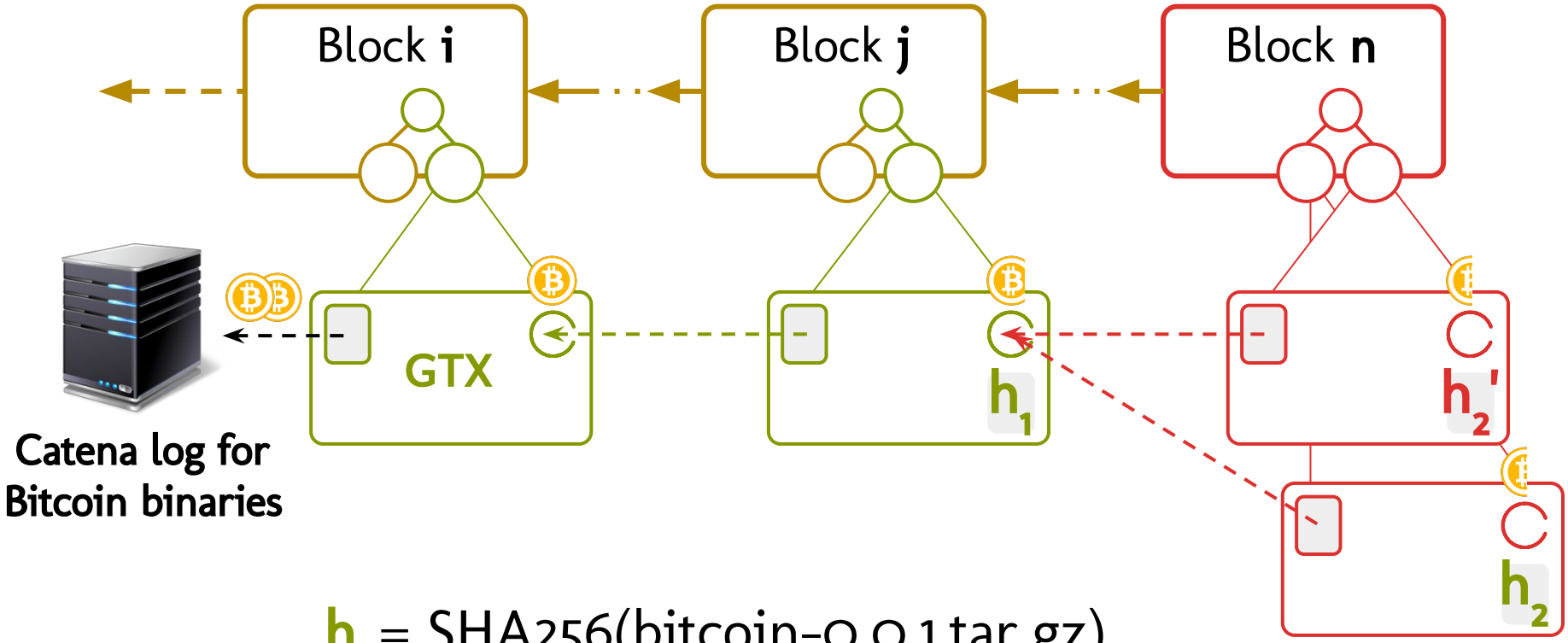
Software transparency to the rescue



h_1 = SHA256(bitcoin-0.0.1.tar.gz)

h_2' = SHA256(evilcoin-0.0.2.tar.gz)

Software transparency to the rescue

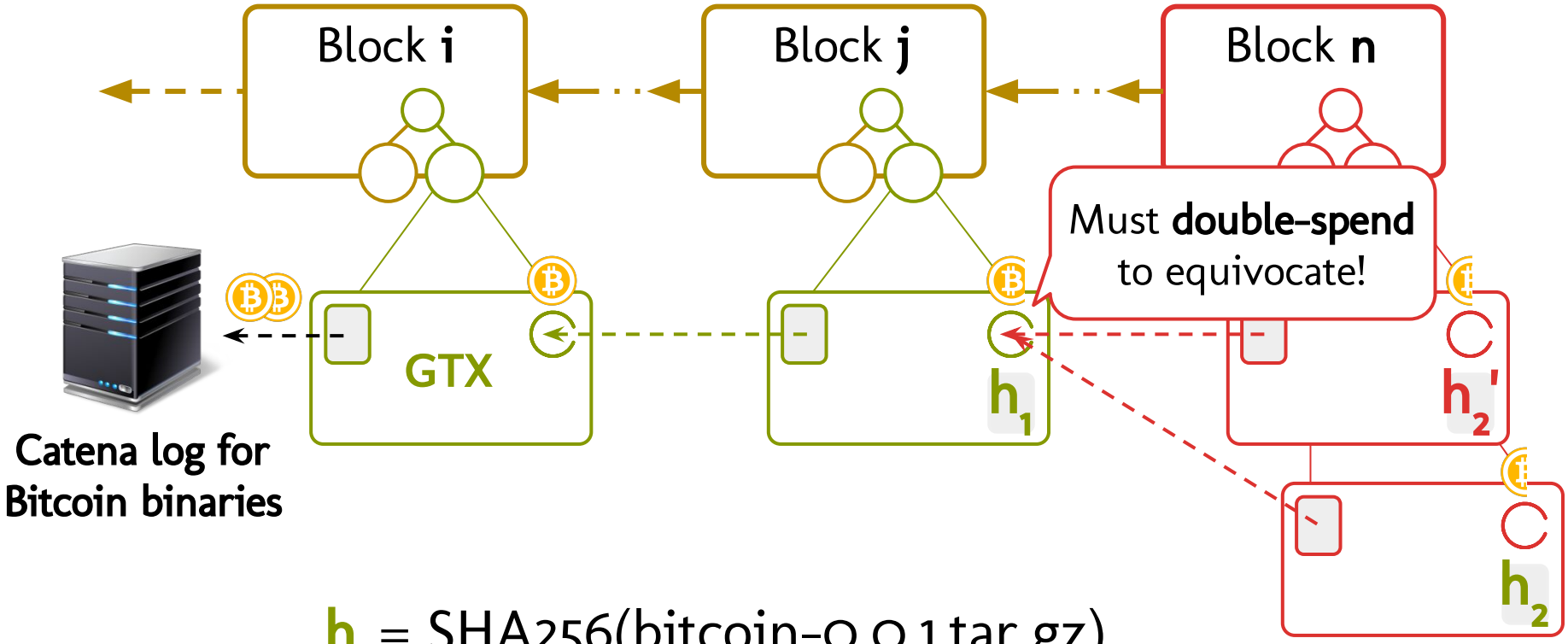


h_1 = SHA256(bitcoin-0.0.1.tar.gz)

h_2' = SHA256(evilcoin-0.0.2.tar.gz)

h_2 = SHA256(bitcoin-0.0.2.tar.gz)

Software transparency to the rescue



h_1 = SHA256(bitcoin-0.0.1.tar.gz)

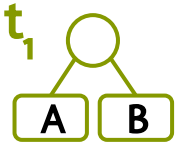
h_2' = SHA256(evilcoin-0.0.2.tar.gz)

h_2 = SHA256(bitcoin-0.0.2.tar.gz)

Overview

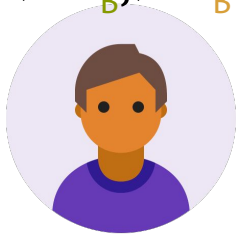
1. What?
2. How?
3. Why?
    a. Secure software update
    b. Secure messaging

Public-key distribution

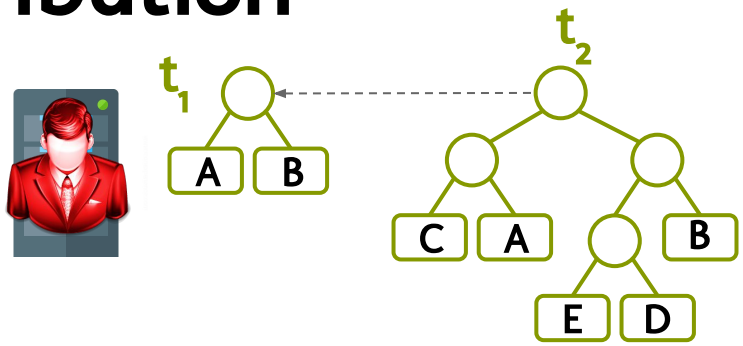


A = {*Alice*, PK_A}, SK_A

B = {*Bob*, PK_B}, SK_B

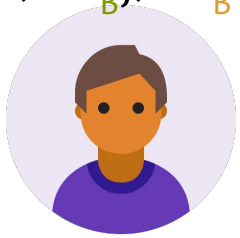


Public-key distribution

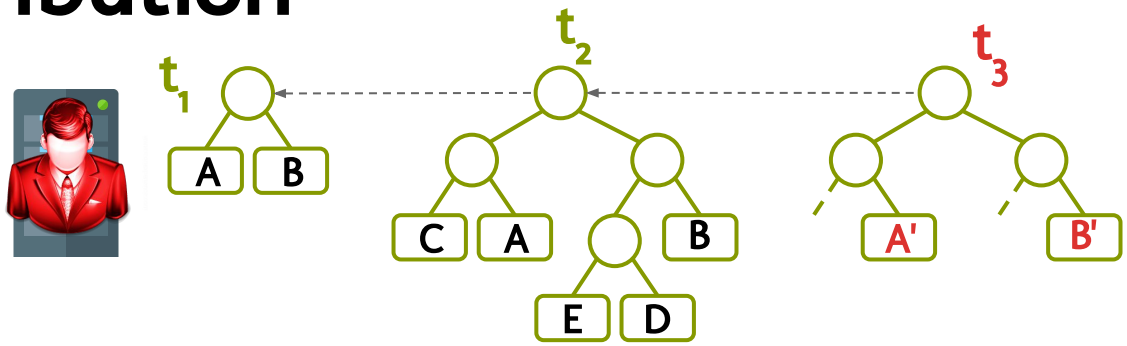


A = {*Alice*, PK_A}, SK_A

B = {*Bob*, PK_B}, SK_B



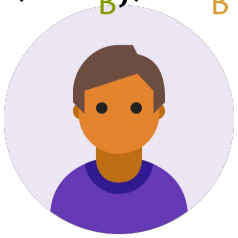
Public-key distribution



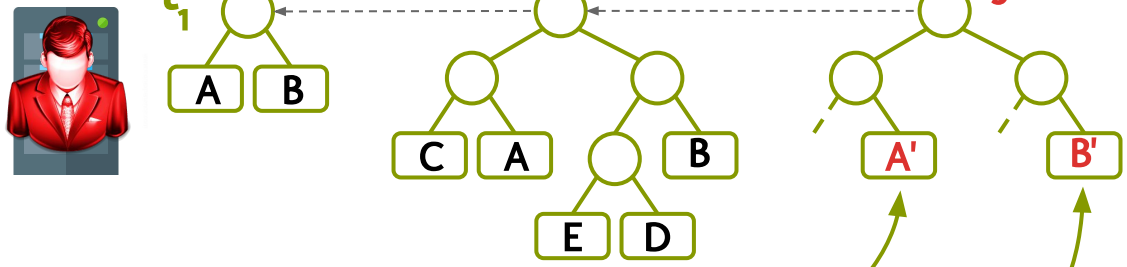
A = {*Alice*, PK_A}, SK_A

A' = {*Alice*, PK_M}, SK_M
B' = {*Bob*, PK_M}, SK_M

B = {*Bob*, PK_B}, SK_B



Public-key distribution



I've been impersonated!

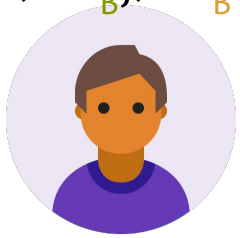
A = {*Alice*, PK_A}, SK_A

A' = {*Alice*, PK_M}, SK_M

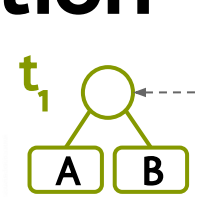
B' = {*Bob*, PK_M}, SK_M

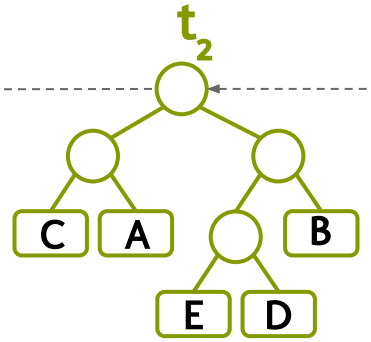
I've been impersonated!

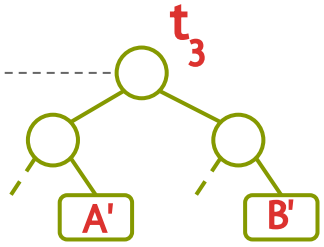
B = {*Bob*, PK_B}, SK_B



Public-key distribution







I've been impersonated!

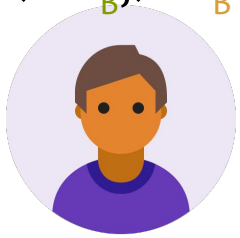
A = {*Alice*, PK_A}, SK_A

A' = {*Alice*, PK_M}, SK_M

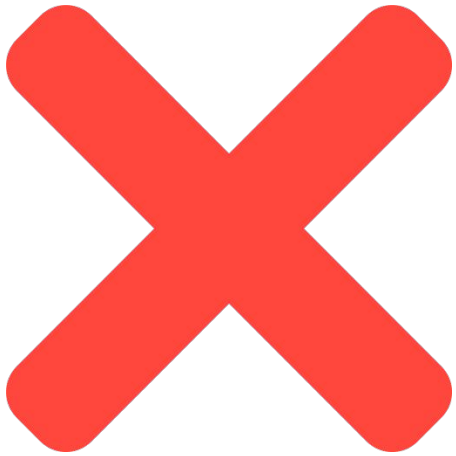
B' = {*Bob*, PK_M}, SK_M

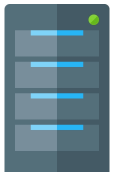
I've been impersonated!

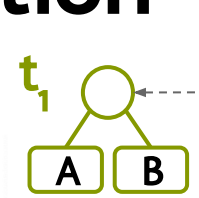
B = {*Bob*, PK_B}, SK_B

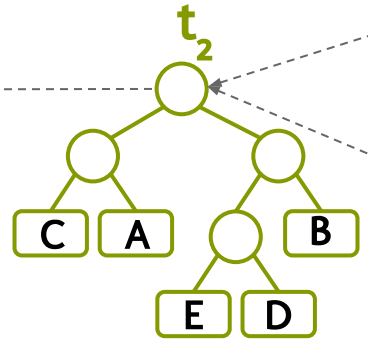


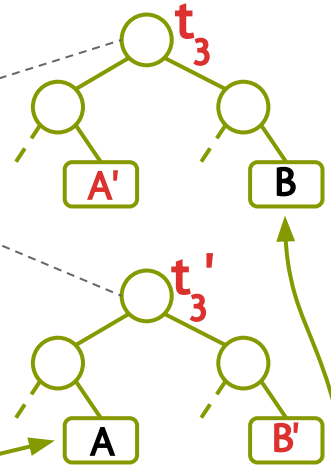
Public-key distribution











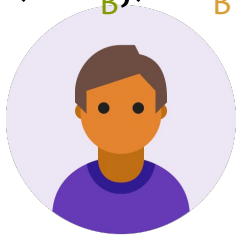
I'm **not** impersonated.

A = {*Alice*, PK_A}, SK_A

A' = {*Alice*, PK_M}, SK_M
B' = {*Bob*, PK_M}, SK_M

I'm **not** impersonated.

B = {*Bob*, PK_B}, SK_B

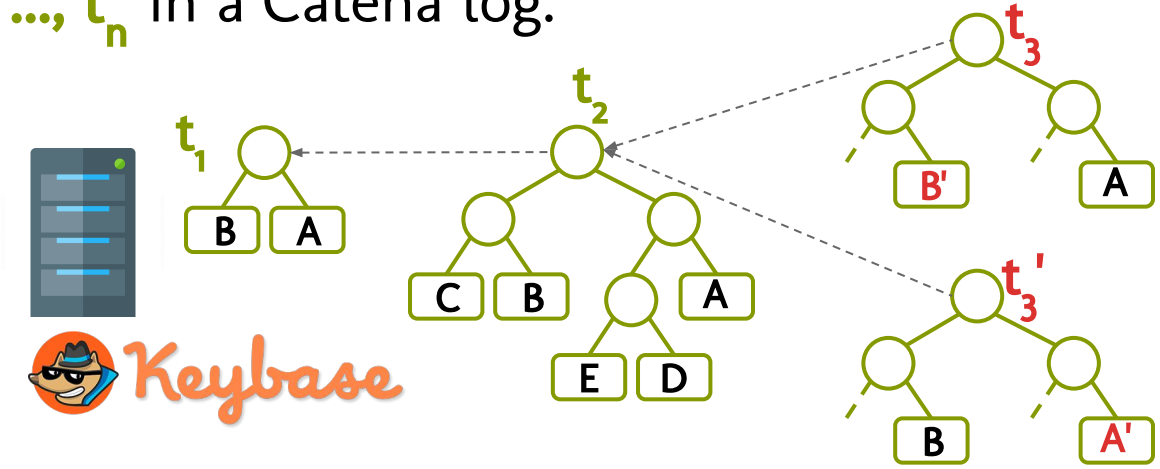


KeyChat

Idea: Store $t_1, t_2, \ldots, t_n$ in a Catena log.

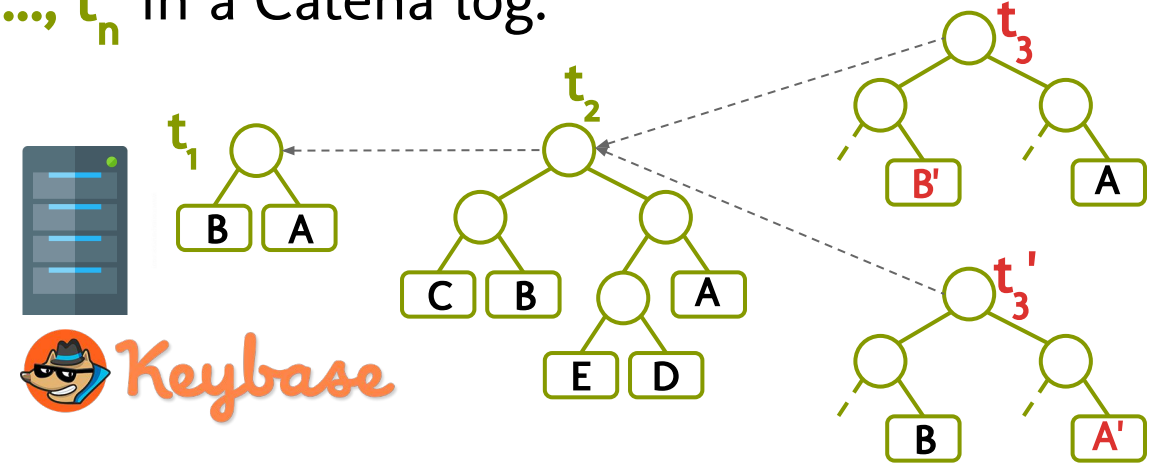
KeyChat

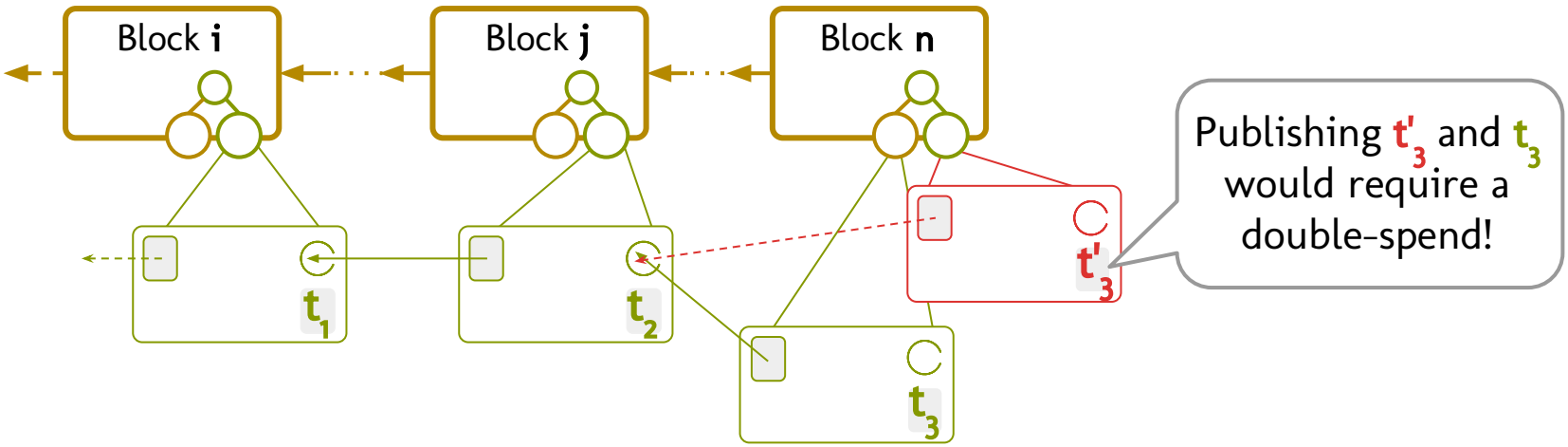
Idea: Store $t_1, t_2, \ldots, t_n$ in a Catena log.



KeyChat

Idea: Store $t_1, t_2, \ldots, t_n$ in a Catena log.





Overview

1. What?
2. How?
3. Why?
    a. Secure software update
    b. Secure messaging
    c. "Blockchain" for X

I need a "blockchain" for ...

I need a "blockchain" for ...

IoT? Self-driving cars? Digital identity? Issue diplomas? Health care? Voting?

I need a "blockchain" for ...

IoT? Self-driving cars? Digital identity? Issue diplomas? Health care? Voting?

"Blockchain" = Byzantine State Machine Replication (SMR)

I need a "blockchain" for ...

IoT? Self-driving cars? Digital identity? Issue diplomas? Health care? Voting?

"Blockchain" = Byzantine State Machine Replication (SMR) =

= Agree on log of ops + Execute ops

I need a "blockchain" for ...

IoT? Self-driving cars? Digital identity? Issue diplomas? Health care? Voting?

"Blockchain" = Byzantine State Machine Replication (SMR) =

= Agree on log of ops + Execute ops = Agree on final state.

I need a "blockchain" for ...

IoT? Self-driving cars? Digital identity? Issue diplomas? Health care? Voting?

"Blockchain" = Byzantine State Machine Replication (SMR) =

= Agree on log of ops + Execute ops = Agree on final state.

Permissioned "blockchain" via:

- Your favorite Byzantine SMR algorithm

I need a "blockchain" for ...

IoT? Self-driving cars? Digital identity? Issue diplomas? Health care? Voting?

"Blockchain" = Byzantine State Machine Replication (SMR) =

= Agree on log of ops + Execute ops = Agree on final state.

Permissioned "blockchain" via:

- Your favorite Byzantine SMR algorithm
- Ethereum Smart Contract (pay fees per Ethereum op)

I need a "blockchain" for ...

IoT? Self-driving cars? Digital identity? Issue diplomas? Health care? Voting?

"Blockchain" = Byzantine State Machine Replication (SMR) =

= Agree on log of ops + Execute ops = Agree on final state.

Permissioned "blockchain" via:

- Your favorite Byzantine SMR algorithm
- Ethereum Smart Contract (pay fees per Ethereum op)
- **Catena + 2f+1 replicas (pay fees per op batch)**

I need a "blockchain" for ...

IoT? Self-driving cars? Digital identity? Issue diplomas? Health care? Voting?

"Blockchain" = Byzantine State Machine Replication (SMR) =

= Agree on log of ops + Execute ops = Agree on final state.

Permissioned "blockchain" via:

- Your favorite Byzantine SMR algorithm
- Ethereum Smart Contract (pay fees per Ethereum op)
- **Catena + 2f+1 replicas (pay fees per op batch)**
- Don't need execution? Use Catena directly.

I need a "blockchain" for ...

IoT? Self-driving cars? Digital identity? Issue diplomas? Health care? Voting?

"Blockchain" = Byzantine State Machine Replication (SMR) =

= Agree on log of ops + Execute ops = Agree on final state.

Permissioned "blockchain" via:

- Your favorite Byzantine SMR algorithm
- Ethereum Smart Contract (pay fees per Ethereum op)
- **Catena + 2f+1 replicas (pay fees per op batch)**
- Don't need execution? Use Catena directly.

Permissionless "blockchain:" Roll your own. But proceed with caution?

Conclusions

Conclusions

What we did:

- Enabled applications to efficiently leverage Bitcoin's publicly-verifiable consensus
  - Download transactions selectively rather than full blockchain
  - ~46 MB instead of gigabytes of bandwidth

Conclusions

What we did:

- Enabled applications to efficiently leverage Bitcoin's publicly-verifiable consensus
  - Download transactions selectively rather than full blockchain
  - ~46 MB instead of gigabytes of bandwidth

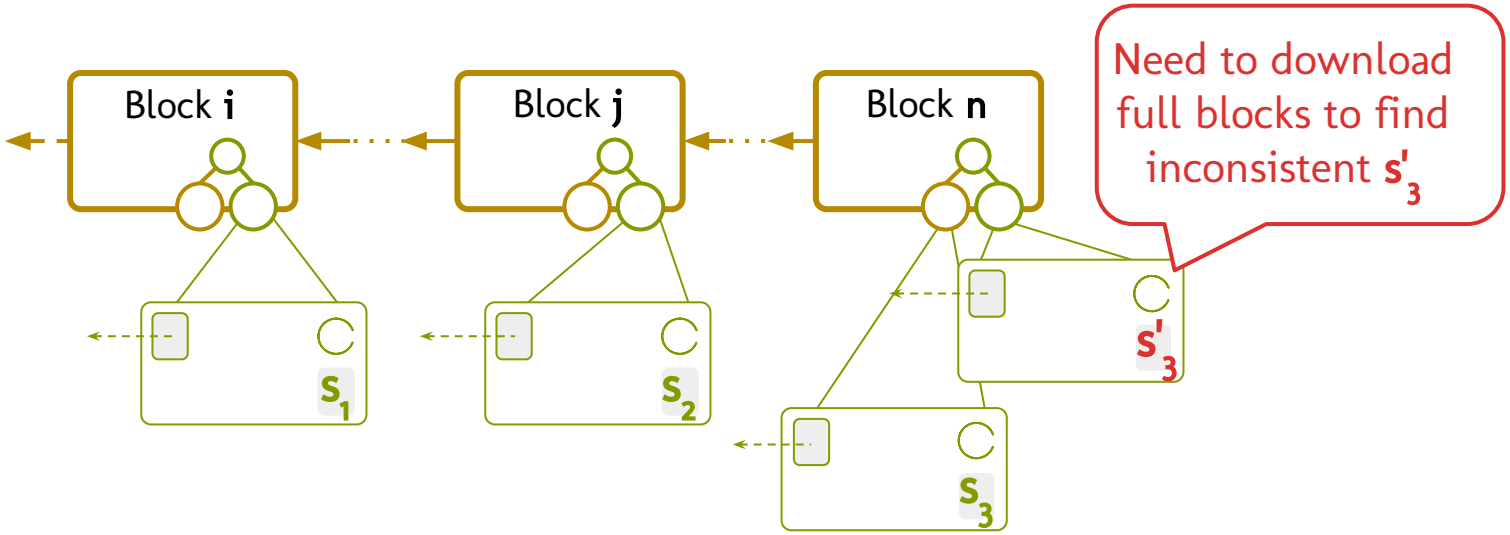
Why it matters:

- Secure software update schemes
- Public-key directories for HTTPS and secure messaging
- "Blockchain" for X

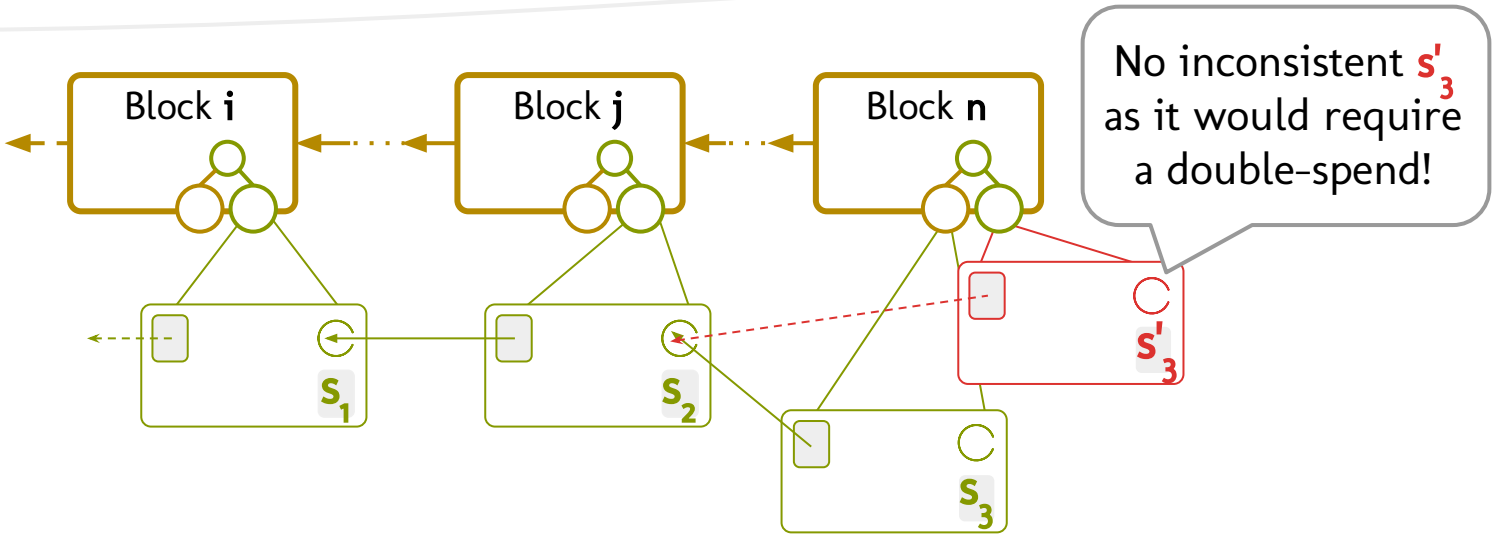
For more, read our paper!

Ask me questions! https://people.csail.mit.edu/alinush/

Previous work



Catena



Cut slides

Public-key distribution

Public-key distribution

PK_A, SK_A

PK_M, SK_M

PK_B, SK_B

Public-key distribution

Public-key distribution

Public-key distribution

Public-key distribution

Public-key distribution

Bob's PK_M

PK_A, SK_A

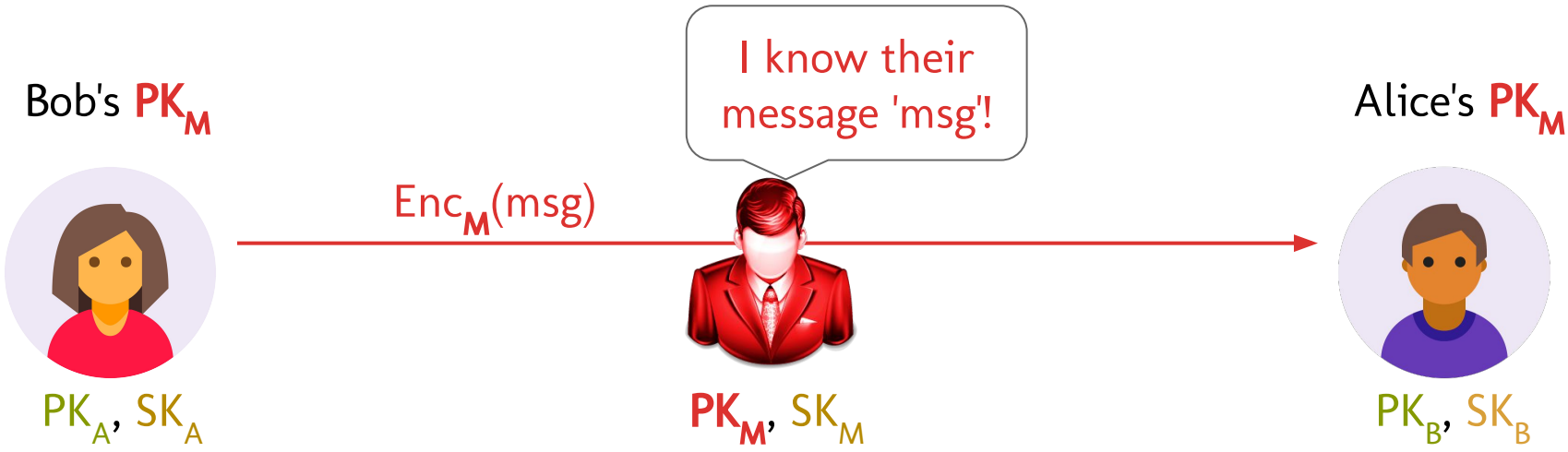
PK_M, SK_M

Alice's PK_M

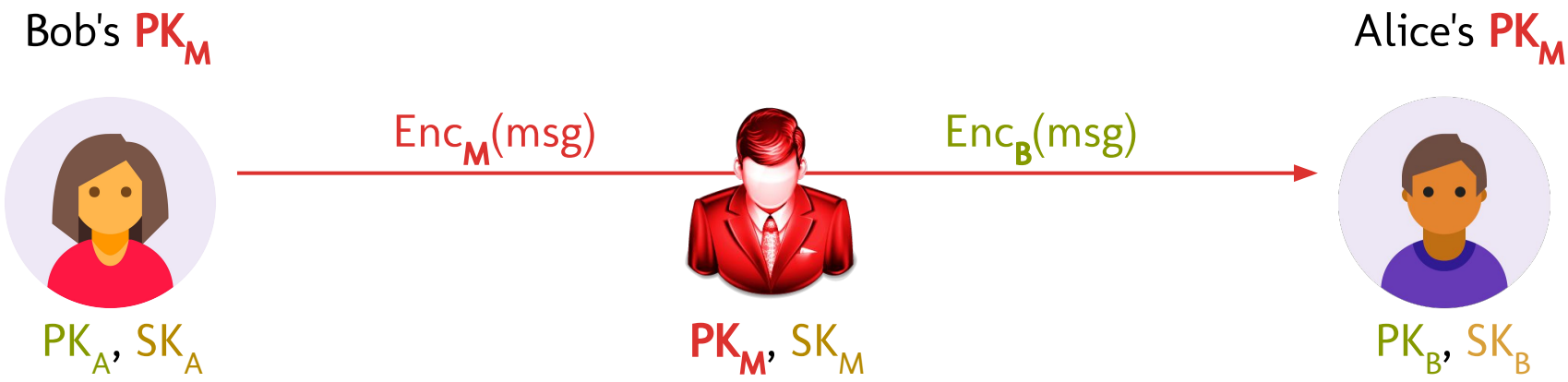
PK_B, SK_B

Public-key distribution

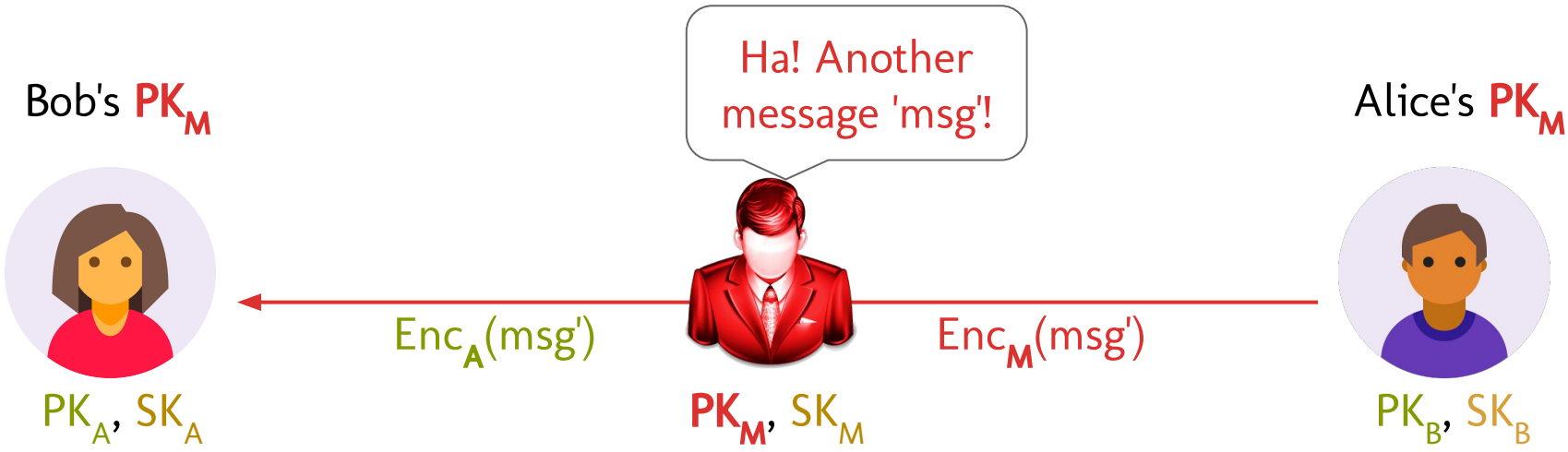
Public-key distribution



Public-key distribution

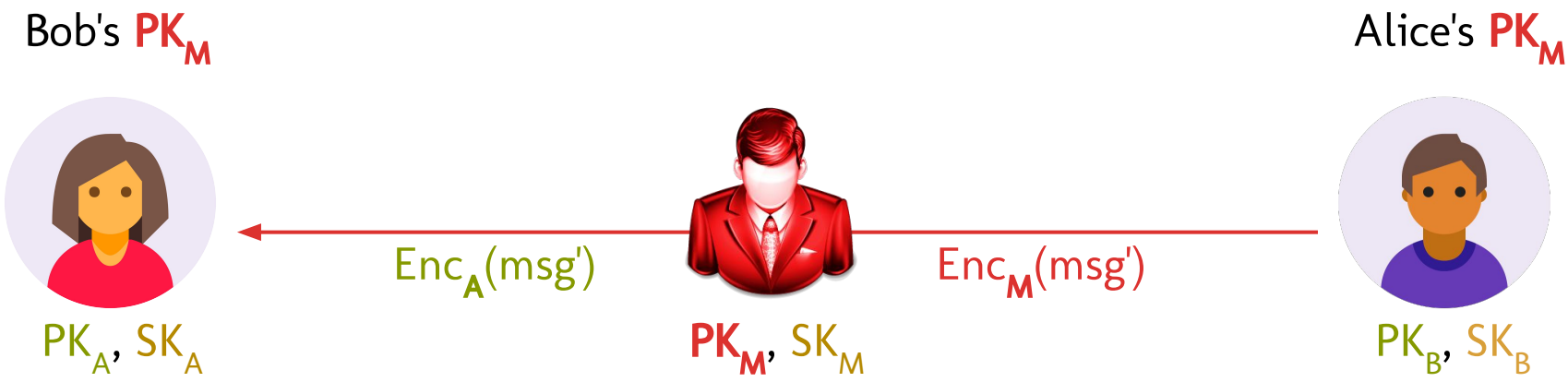


Public-key distribution



Public-key distribution

Using Certificate Authorities (CAs) does not help much: Mallory compromises or coerces CA instead.



Extra slides

What is non-equivocation?

- At time **i**, publishes *digest* s_i

Public-key directory

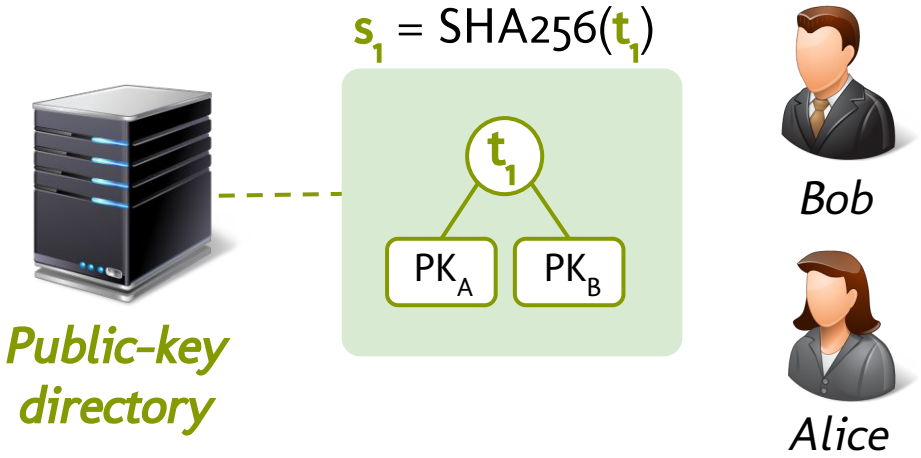
What is non-equivocation?

- At time **i**, publishes a **single** *digest* s_i

Public-key directory

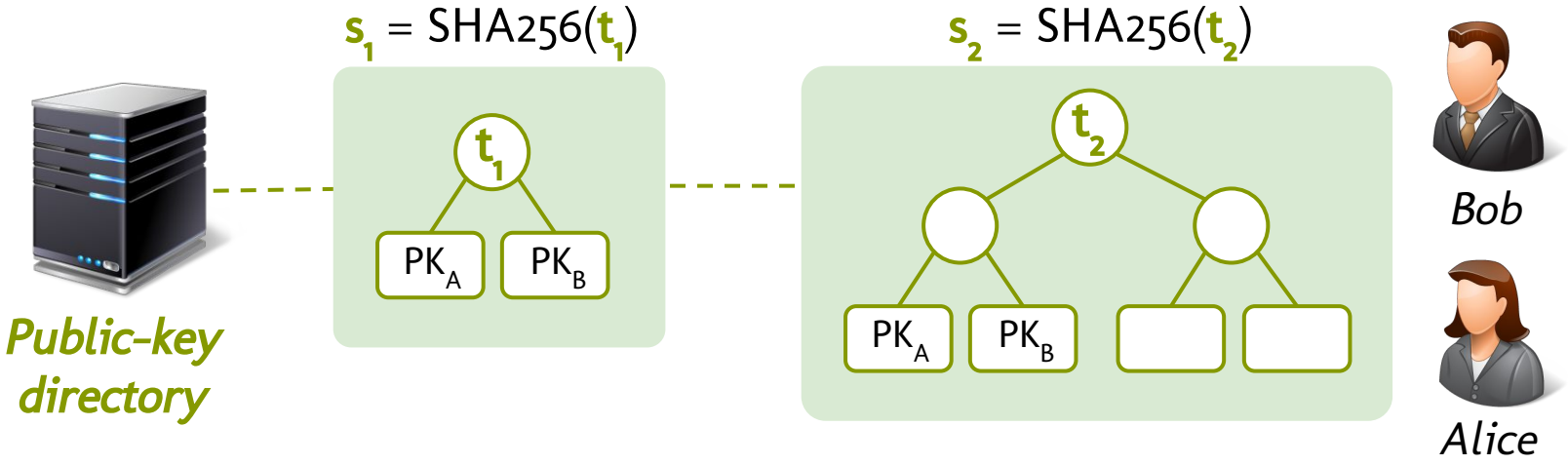
What is non-equivocation?

- At time **i**, publishes a **single** *digest* s_i
- At time **1**, Alice, Bob and others "see" s_1



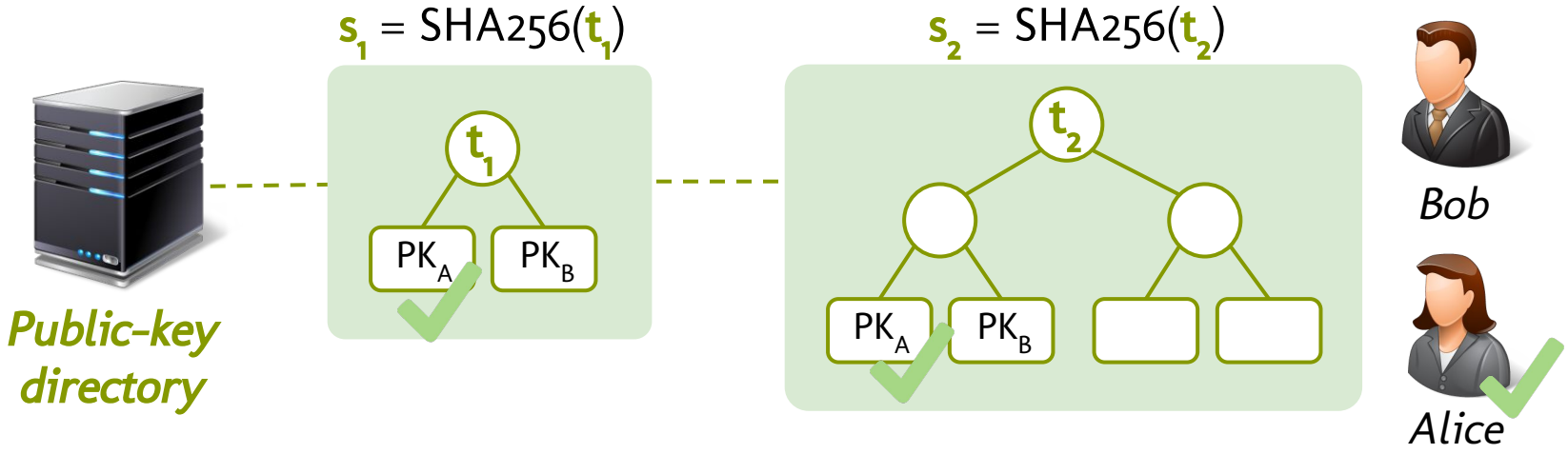
What is non-equivocation?

- At time 2, Alice, Bob and others "see" s_1 , s_2 , ...



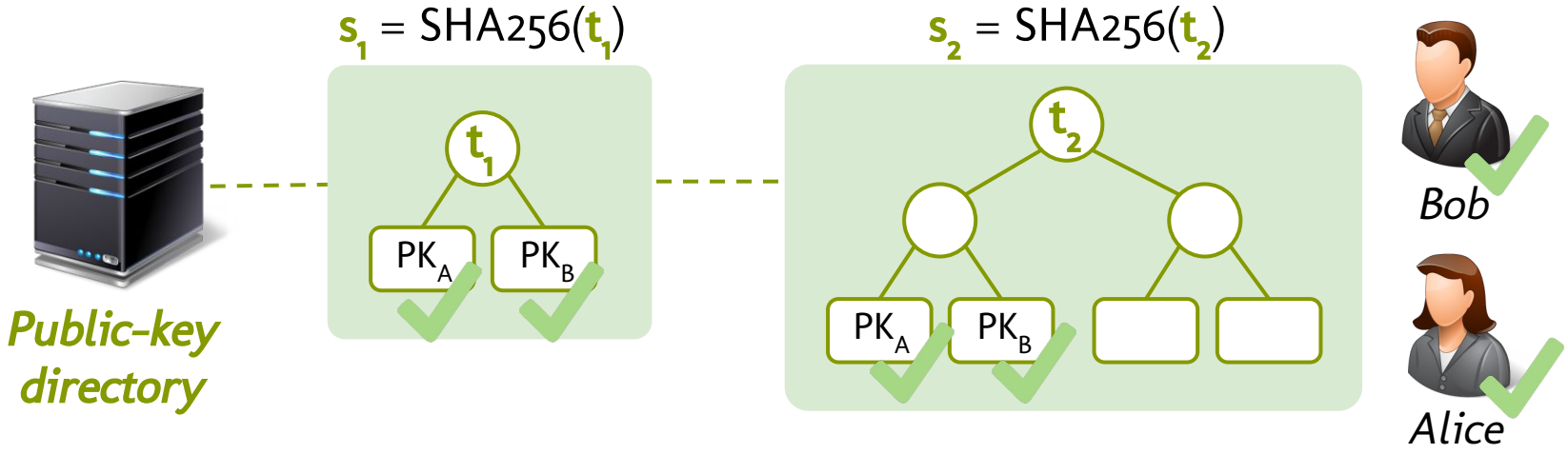
What is non-equivocation?

- Alice and Bob can "monitor" own PKs



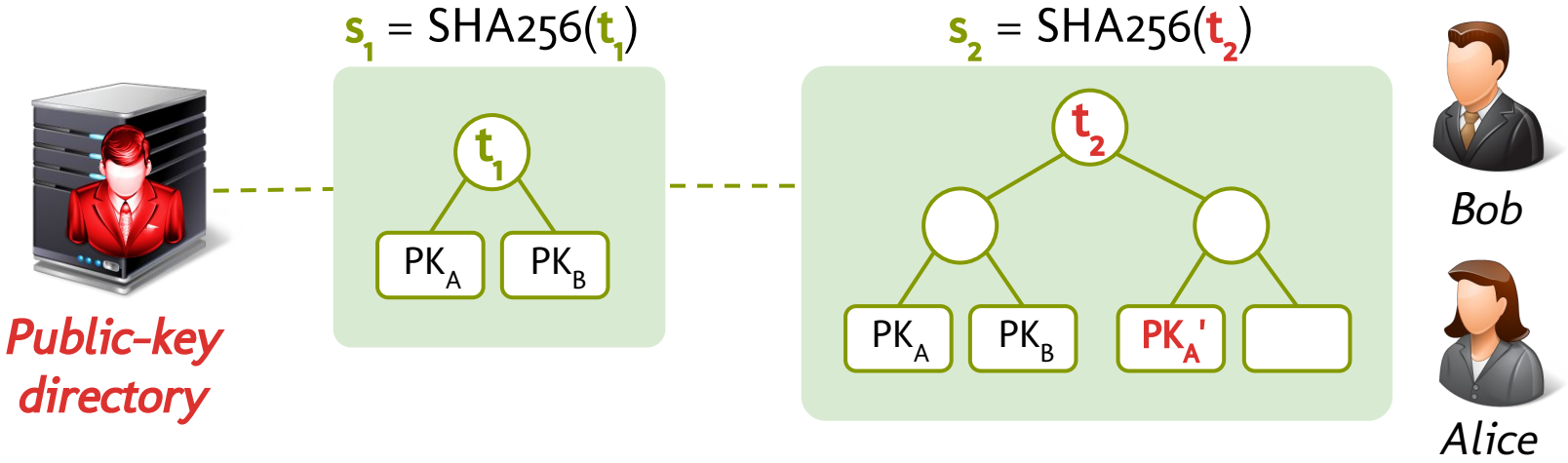
What is non-equivocation?

- Alice and Bob can "monitor" own PKs



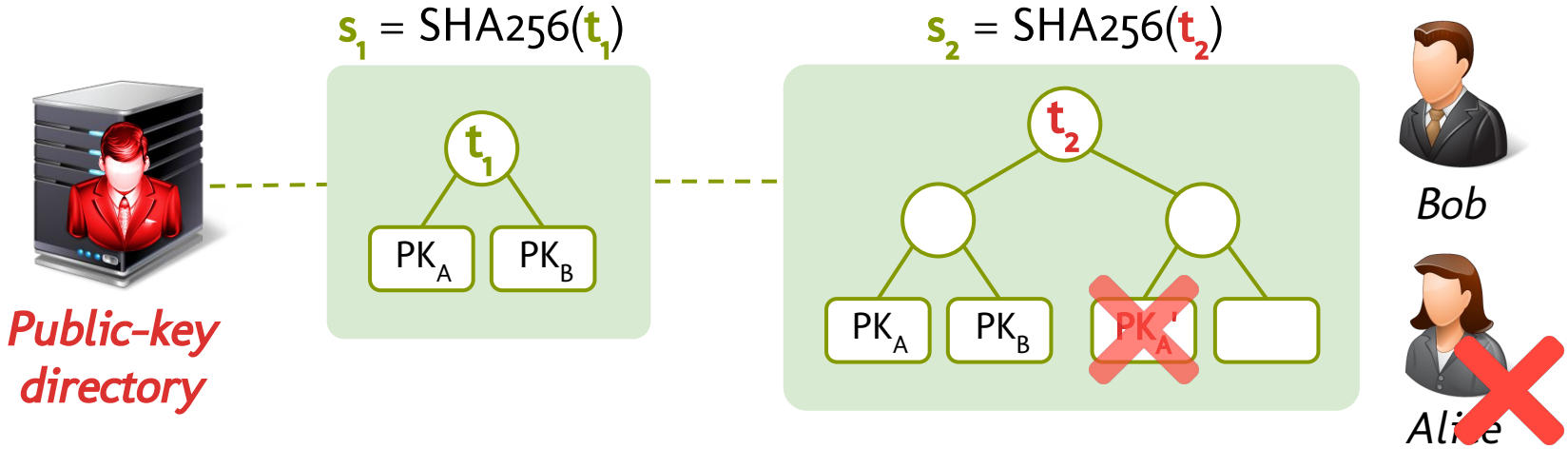
What is non-equivocation?

- Alice and Bob can "monitor" own PKs
- ...and **server** has to impersonate in plain sight



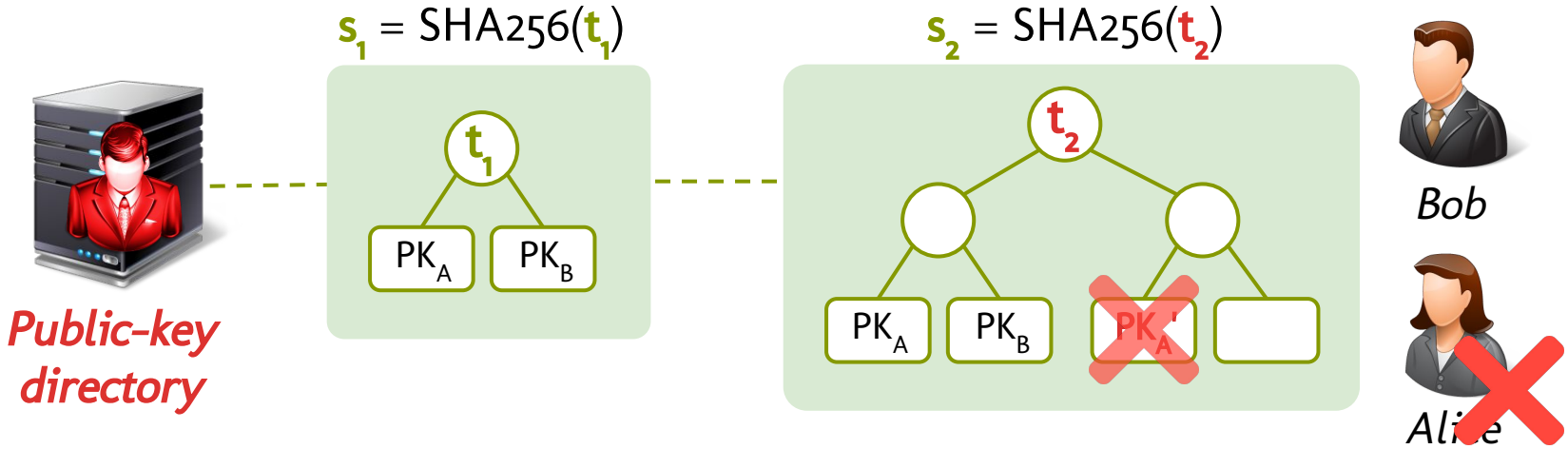
What is non-equivocation?

- Alice and Bob can "monitor" own PKs
- ...and **server** has to impersonate in plain sight



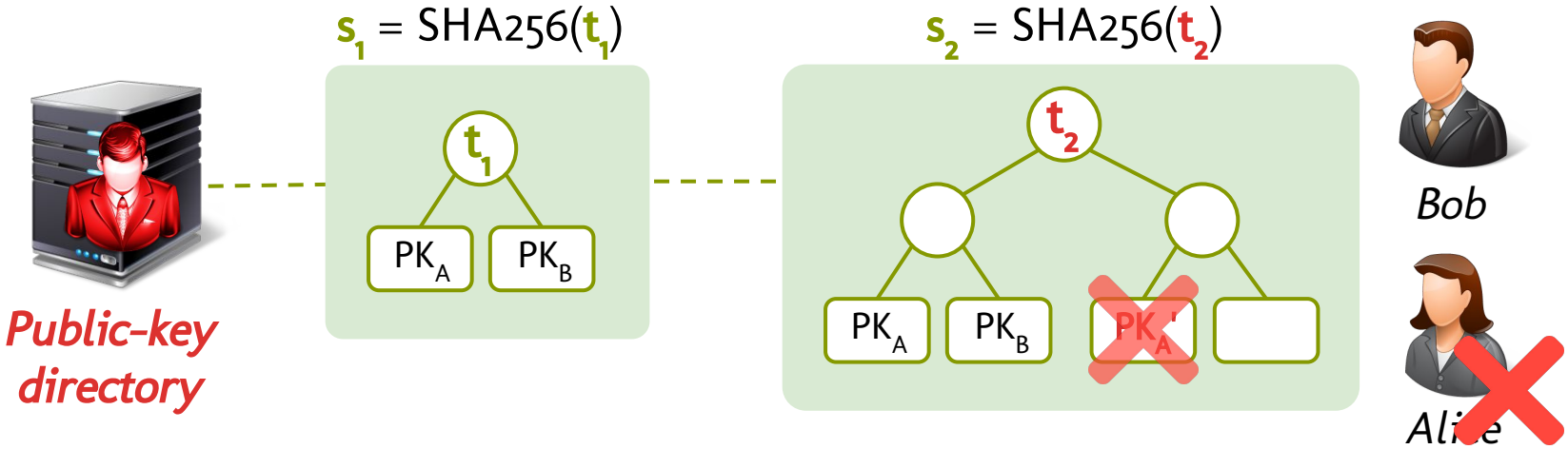
What is non-equivocation?

Good: *"Stating the same thing to all people."*



What is non-equivocation?

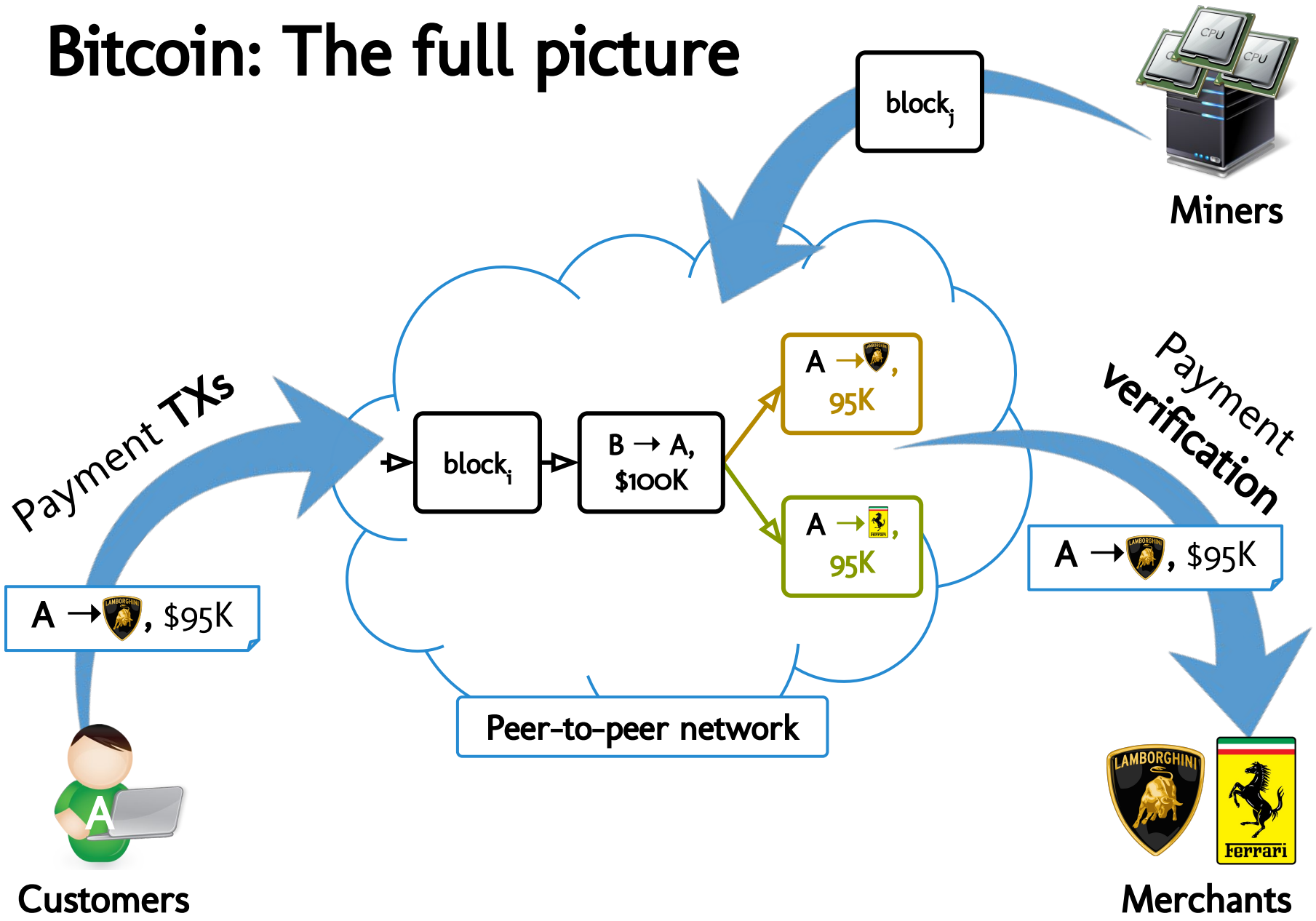
Good: *"Stating the same thing to all people."*



Including statements that are

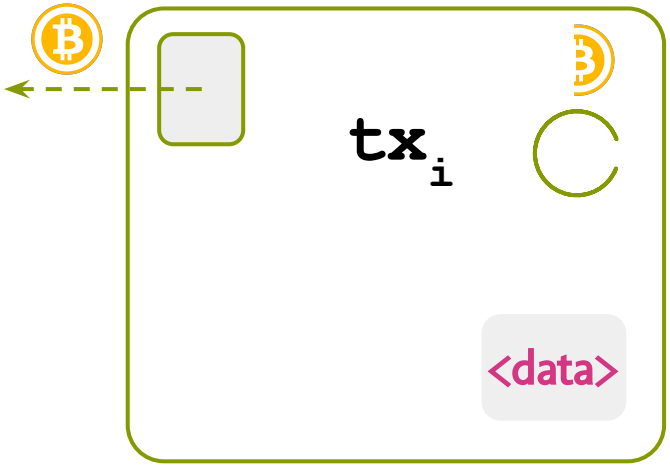
incorrect at the application-layer

Bitcoin: The full picture



Catena transaction format

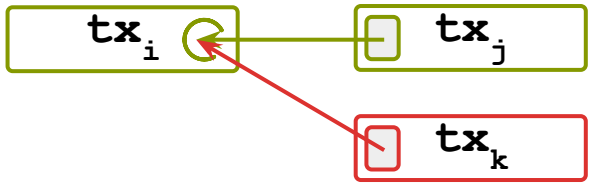
Coins from server for paying TX fees (digital signature)



"Change" coins back to server (public key)

Unspendable `OP_RETURN` output with arbitrary data

A **single spendable output** ⇒ No forks

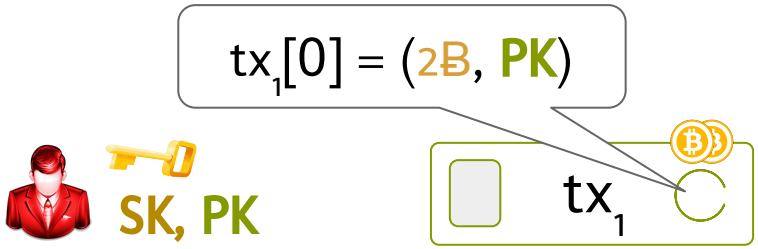


Previous work

"Liar, liar, coins on fire!" (CCS '15)

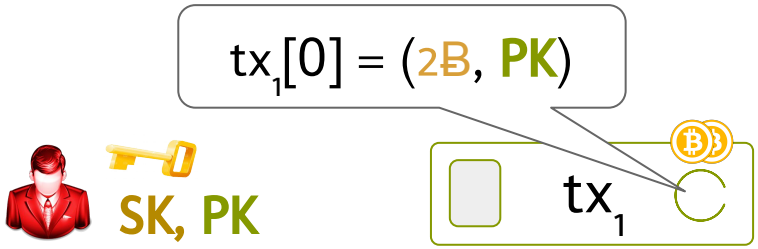
Previous work

"Liar, liar, coins on fire!" (CCS '15)



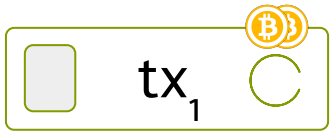
Previous work

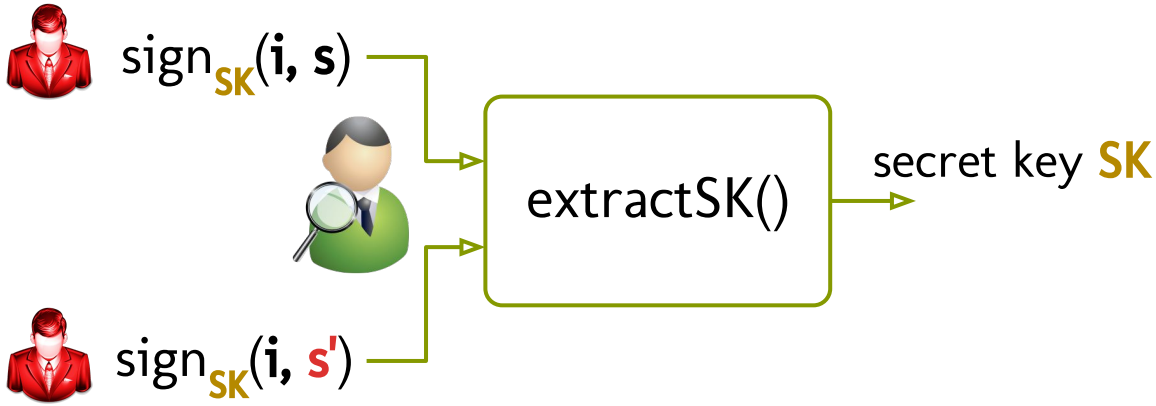
"Liar, liar, coins on fire!" (CCS '15)



Previous work

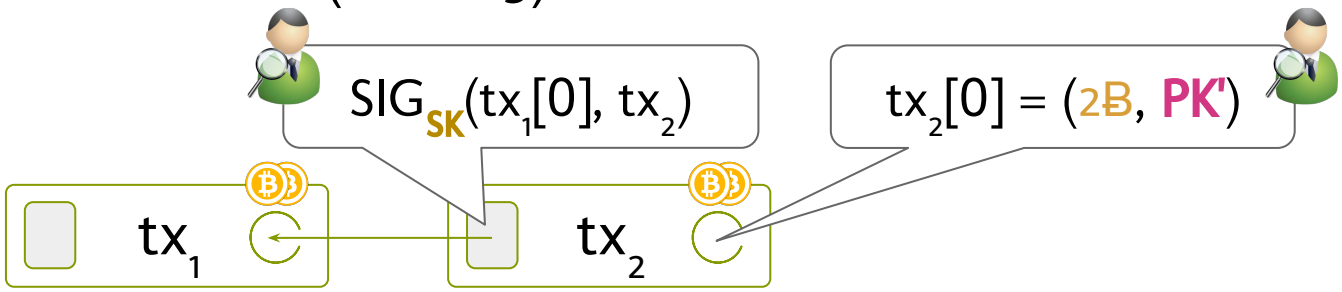
"Liar, liar, coins on fire!" (CCS '15)

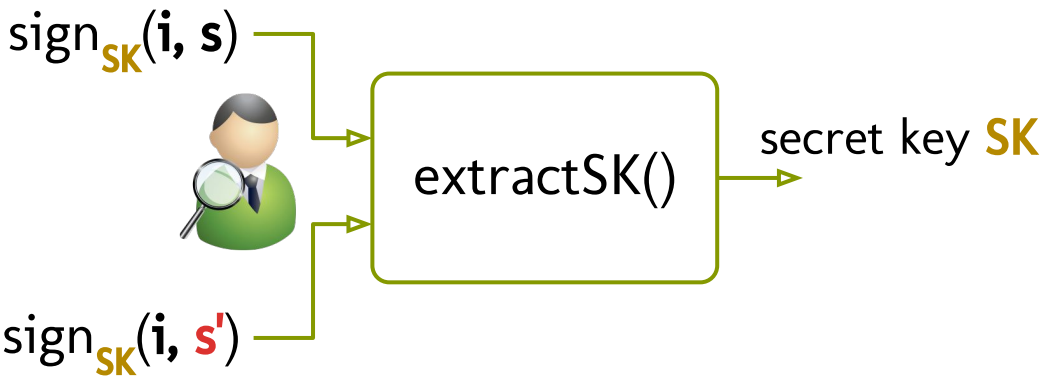




Previous work

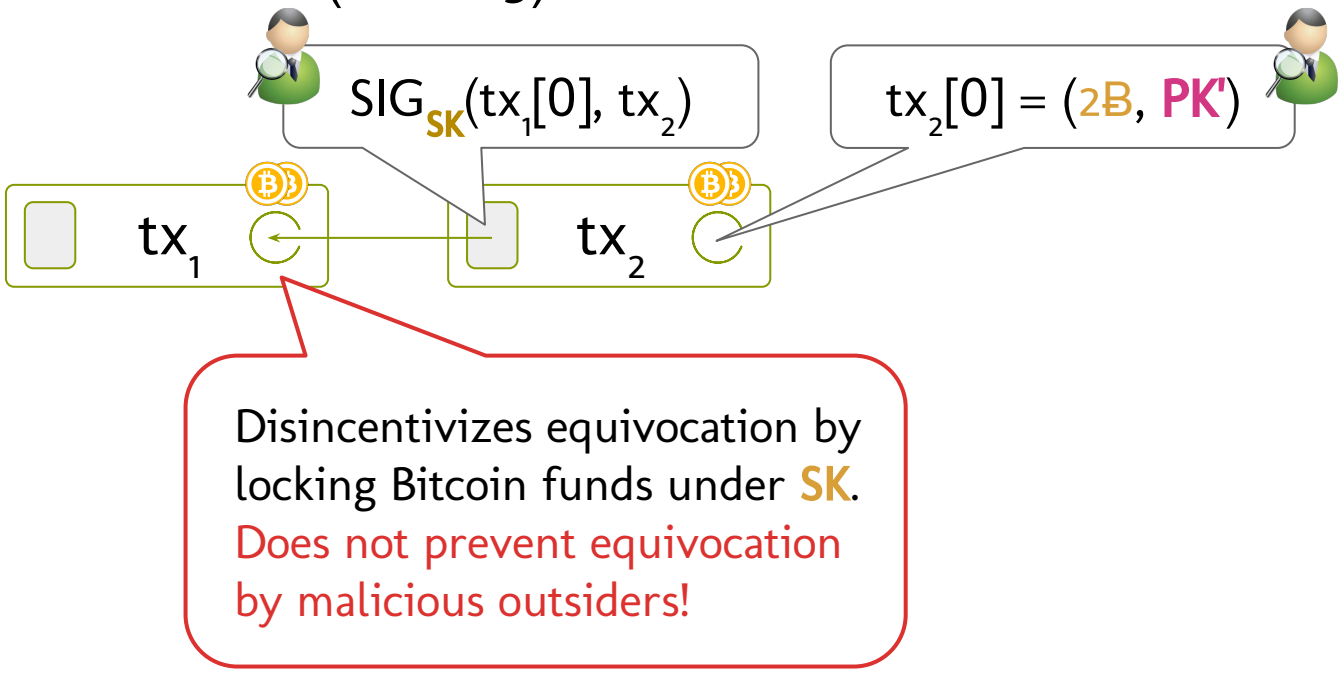
"Liar, liar, coins on fire!" (CCS '15)





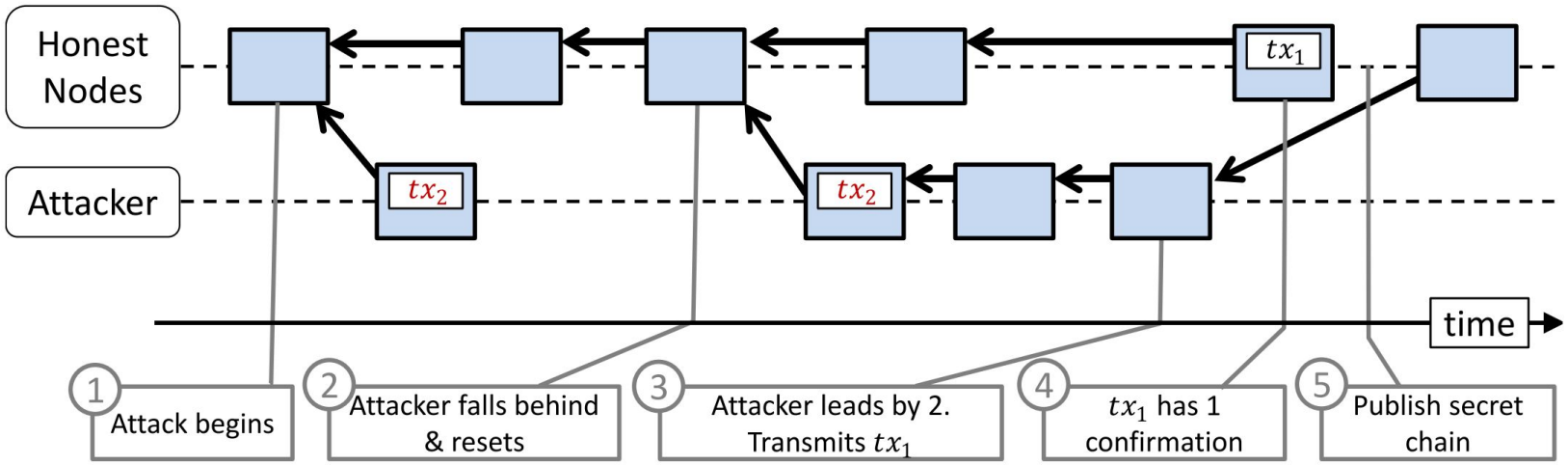
Previous work

"Liar, liar, coins on fire!" (CCS '15)



Source: "Bitcoin's Security Model Revisited," Sompolinsky, Yonatan; Zohar, Aviv

Pre-mining attacks

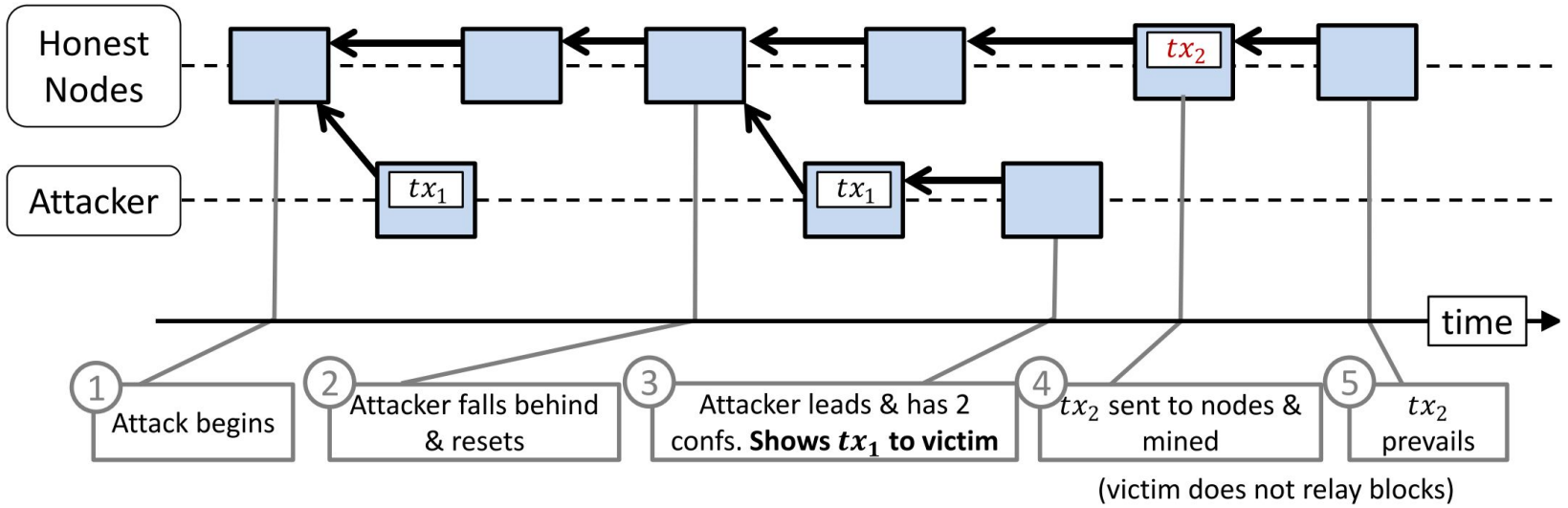


As the attack begins the attacker starts working on a secret chain with tx_2 inside its first block (1). If the attacker's chain is shorter than the honest nodes', the attacker gives up and restarts the attack (2). The attacker manages to gain a lead of 2 blocks (3). He then transmits the transaction he wishes to double spend which is included in a block (4). The transaction now has enough confirmations (1-conf) and the attacker collects his rewards. He then publishes his secret chain and successfully double spends (5). Notice that once the pre-mining stage is concluded, the attack succeeds with probability 1, so miners that see tx_1 that is only broadcast then will always lose the funds.

Fig. 1. The progression of a pre-mining attack on a 1-confirmation defender

Source: "Bitcoin's Security Model Revisited," Sompolinsky, Yonatan; Zohar, Aviv

Generalized "Vector76" attack



As the attack begins the attacker starts working on a secret chain with tx_1 inside its first block (1). If the attacker's chain is too far behind it may restart the attack (2). The attacker manages to gain a lead of 1 blocks, but has the two confirmations on his tx_1 needed to convince the victim (3). He then reveals the secret chain to the victim (that does not relay it), and collects an item in exchange. He then transmits the double spending transactions tx_2 to the network which is then included in a block (4). The network continues to mine atop tx_2 and it eventually prevails (5).

Fig. 3. The progression of a generalized Vector76 attack on a 2-confirmation defender