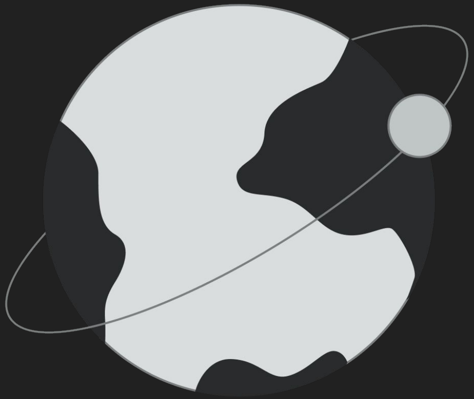
UniRep

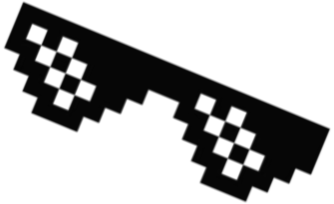
Universal Reputation



Ethereum Foundation’s
Privacy and Scaling Explorations Team

Consequence in Private applications

- **Can we have privacy and consequnces for actions ?**
- **Example: Autis.im Semaphore message board**
  - Every day someone posts 1000 messages about some ico token
  - We don’t know its all comming from the same person
  - But figure its a spam
- **Not attributable so can’t punish**

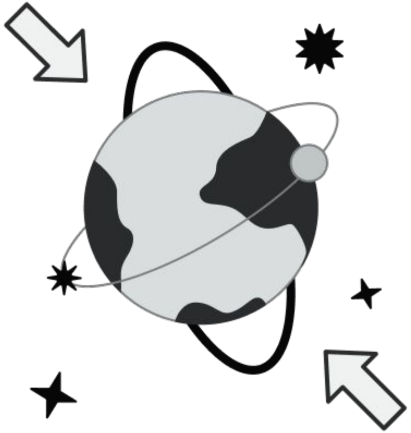


Attribution and privacy are hard

- **We can do objectinve things inside ZKP**
  - Example: There were 10 posts in the last hour
  - Like you can’t send more than x messages per second
- **Subjective things we can’t automatically check**
  - Example: This post is good
  - Instead we need to build social infrastructure to check this

If we could make a private reputation system that would be very powerful.

- Can be sent positive and negative reputation
- Can’t hide positive or negative reputation
- Private data lets you make all kinds of proof about various reputation
- We can build social media
- Non collateralized loans
- Remove capthas and replace with ZKPs



Introduction of Unirep

airb b

B king.com

r ddit

am zon

Cross-App
Reputation System

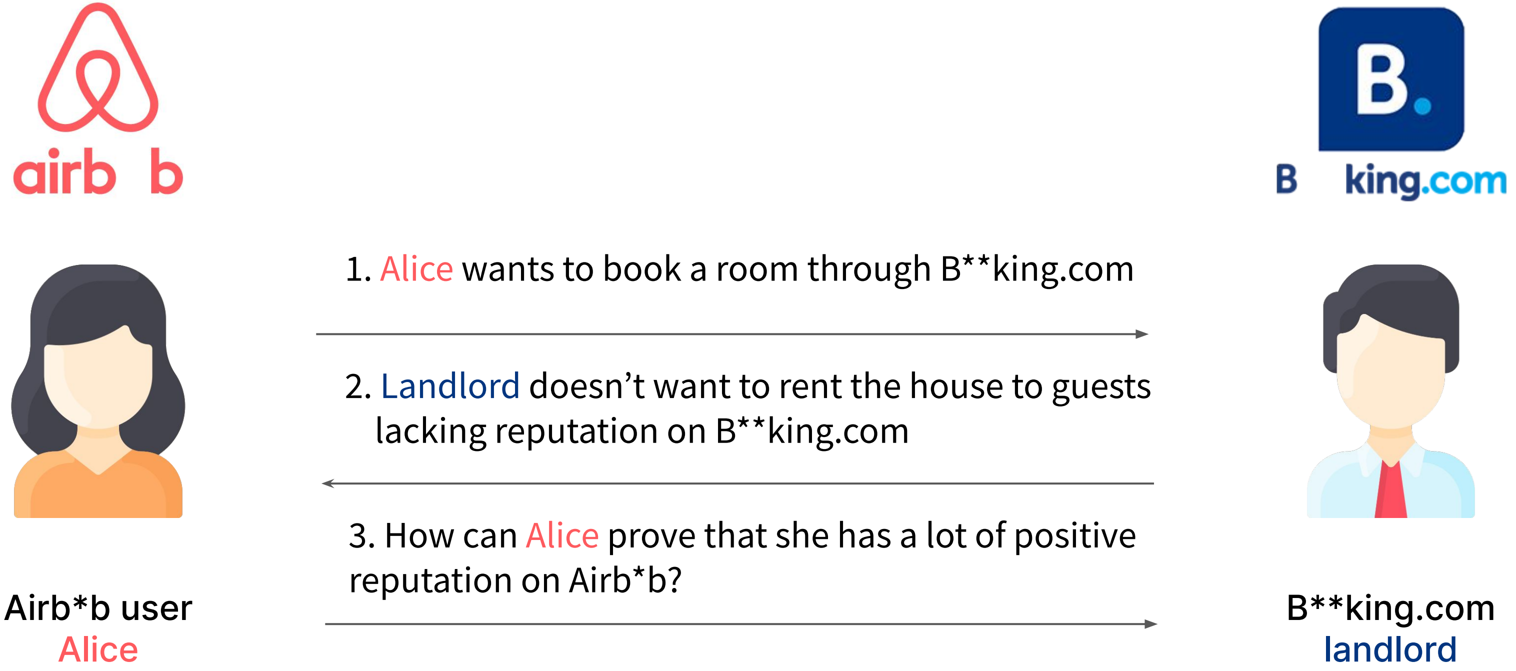
YouTub

Li ked in

T itter

GitH b

Introduction of Unirep



1. Alice wants to book a room through B**king.com

2. Landlord doesn’t want to rent the house to guests lacking reputation on B**king.com

3. How can Alice prove that she has a lot of positive reputation on Airb*b?

Airb*b user
Alice

B**king.com
landlord

Introduction of Unirep

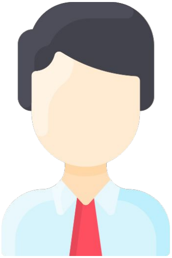
Airb*b user
Alice

How can Alice prove that she has a lot of positive reputation on Airbnb?

e.g. Alice takes a screenshot

- It compromises Alice's privacy
- Screenshot can easily be forged
- Landlord cannot be sure that Alice did not forge the screenshot

Unirep can solve the problem



B**king.com
landlord

Introduction of Unirep

Universal Reputation

- A **private** and **non-repudiable** repuation system.
- Users can receive positive and negative reputation from attesters
- Voluntarily prove that they have at least certain amount of reputation without revealing the exact amount.
- Users cannot refuse to receive reputation from an attester.

Introduction of Unirep

Actors in Unirep:

Attesters

- non-anonymous
- Represent users to give reputation

Users

- anonymous
- Receive reputation
- Prove reputation

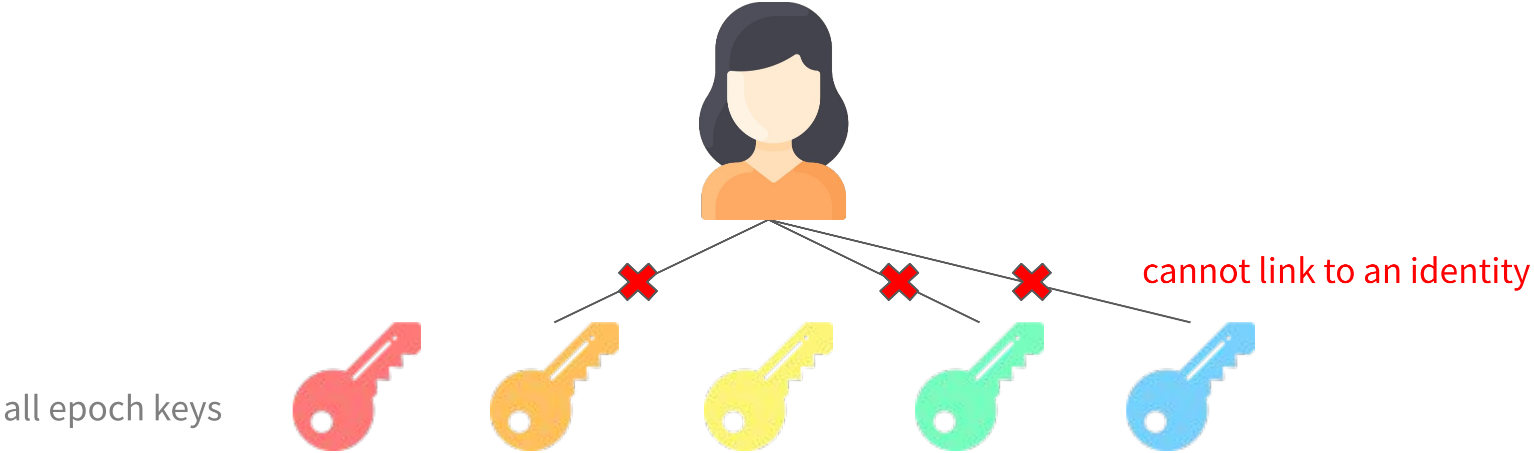
Airb*b user
Alice

B**king.com
landlord

Introduction of Unirep

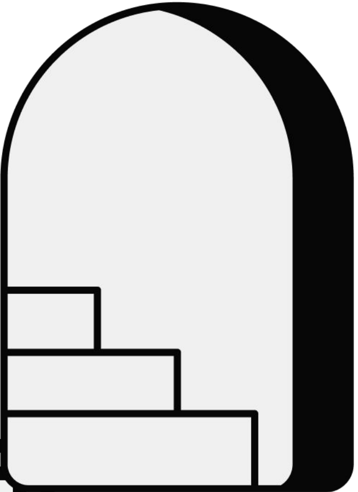
Privacy:

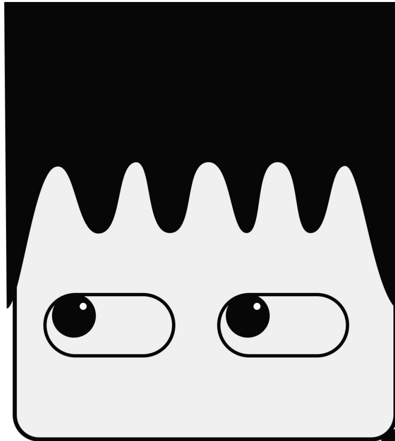
- User uses a **temporary identity** to receive reputation, called an ***epoch key***.
- User can generate *k* epoch keys within an ***epoch*** (e.g. 7 days).
- User can receive all reputation given to these *k* epoch keys.



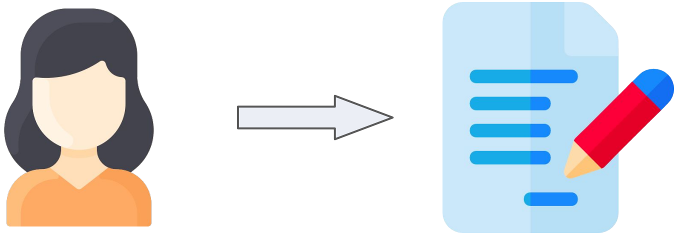
Unirep Social Demo

unirep.social





Unirep Protocol



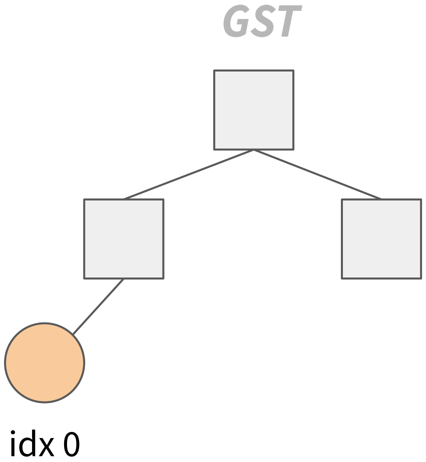
User signs up

1. Generate an identity and an identity commitment through **semaphore**
2. Call the smart contract with the identity commitment
3. Smart contract computes the global state tree leaf

```
hash(idCommitment, defaultUserStateTreeRoot)
```

4. Update the global state tree

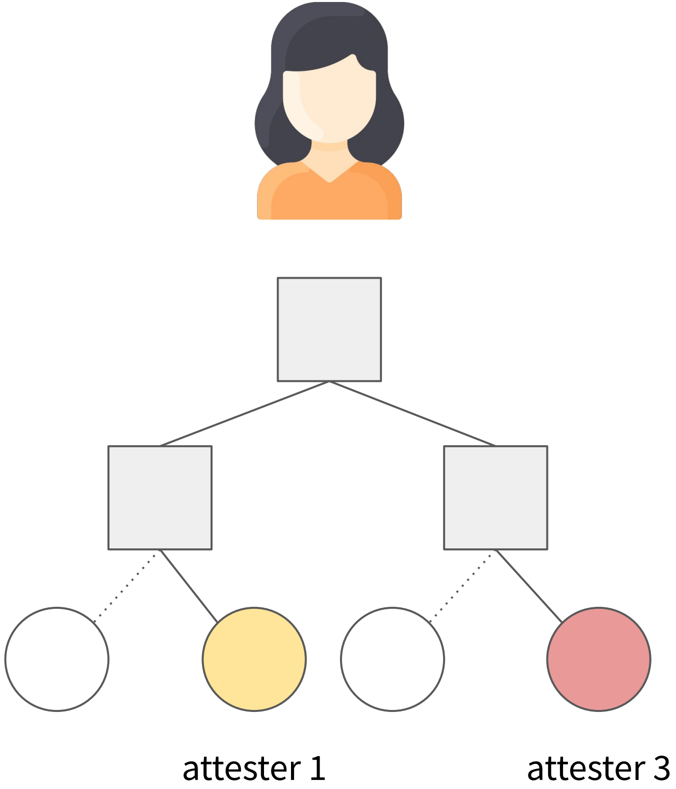
   Insert a leaf in the global state tree



`hash(idCommitment, defaultUserStateTreeRoot)`

User State Tree (UST)

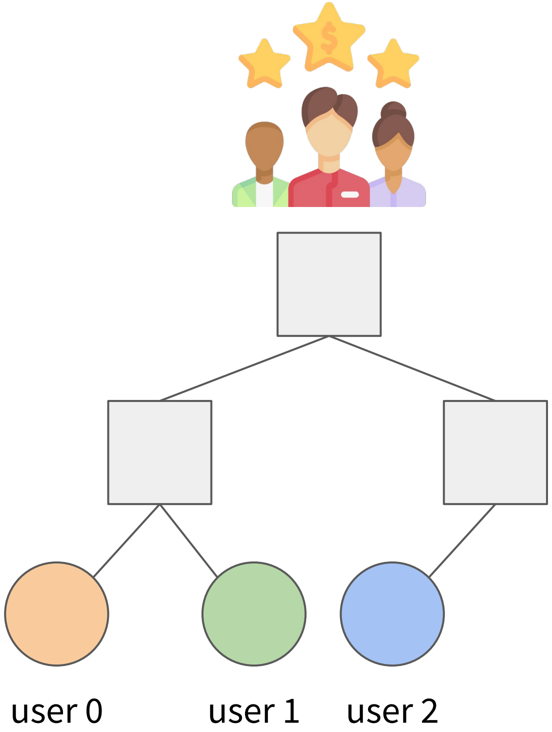
- A sparse merkle tree
- Each user maintains his own user state tree (private data)
- *Leaf ID:* reputation from an attester id
  Leaf value: accumulated reputation from the attester
  `hash(posRep, negRep, graffiti, signUpFlag)`



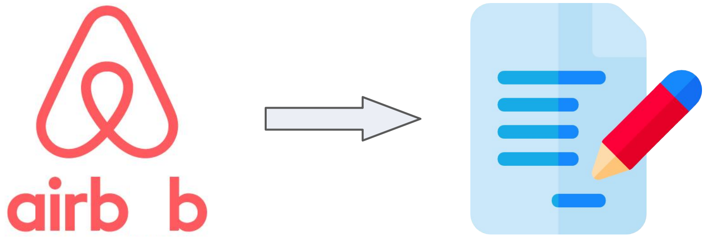
Global State Tree (GST)

- A incremental merkle tree
- All users share a global state tree (public data)
- *Leaf value:* user sign-up state/ user transitioned state

  `hash(id, userStateTreeRoot)`



Unirep Protocol



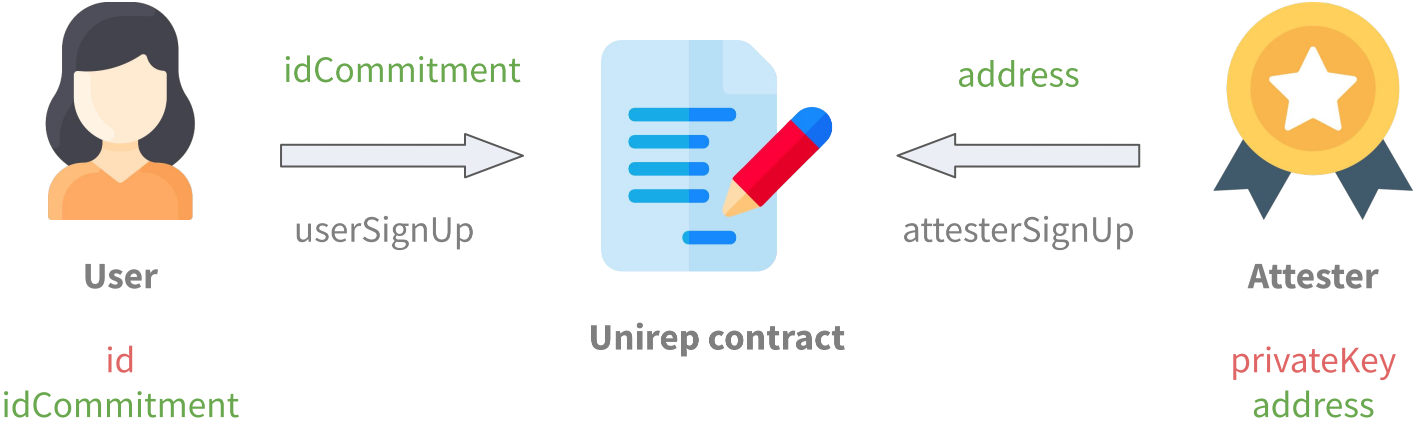
Attester signs up

1. Call the Unirep smart contract with
   a. the attester’s ethereum wallet or
   b. another smart contract
2. The Unirep contract maps the attester’s address to an attester id

```
attester[address] = attesterId
```

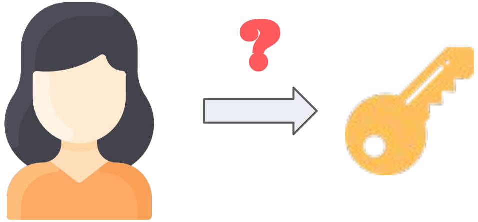
Unirep Protocol

User/ attester signs up



Unirep Protocol

User generates an epoch key to receive reputation



- Epoch key is computed by

```
hash(identity, epoch , nonce)
```

User can choose nonce from 0 to (k-1) to have k epoch keys per epoch

How can a user ensure that the owner of the epoch key has signed up?

How can a user ensure that the epoch key is not a random number?

User generates an epoch key with a ZK proof

Unirep Protocol

What is inside an epoch key proof?

- public input:

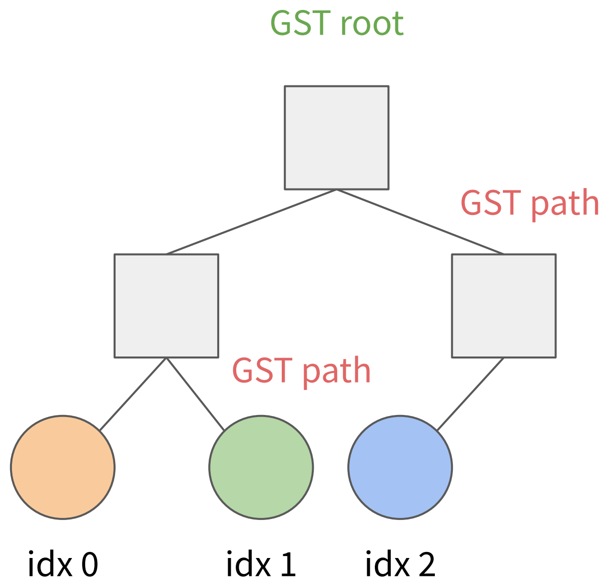
  epoch key , epoch, GST root

- private input:

  identity, epoch key nonce,

  UST root, GST path

- constraints:



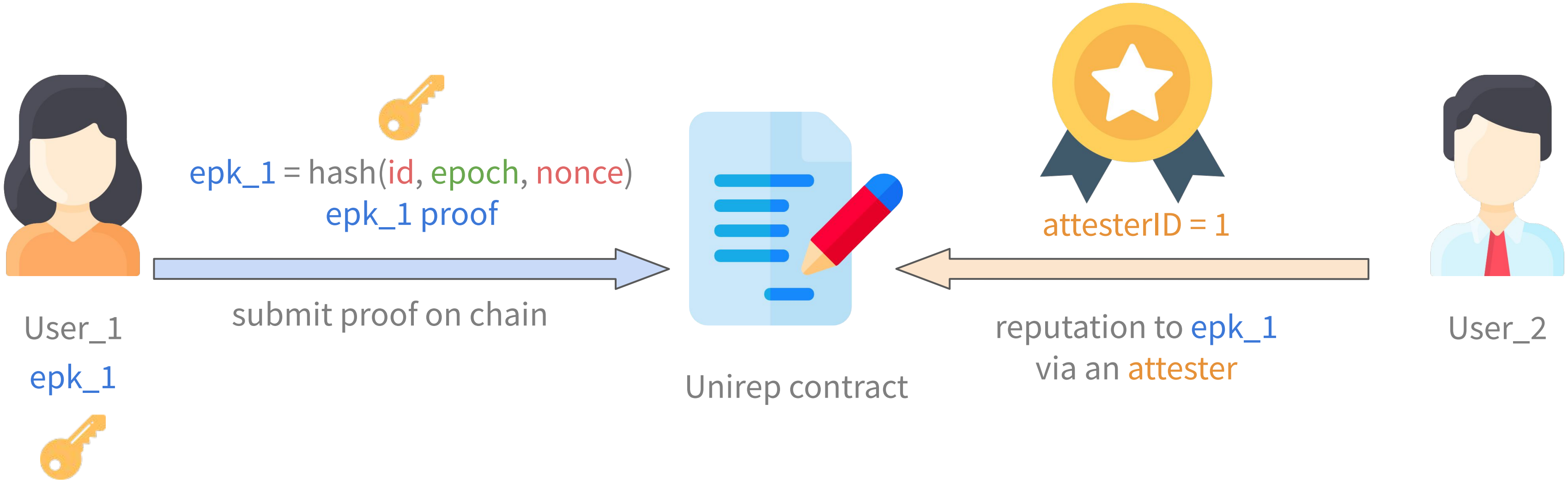
1. Check if user exists in the Global State Tree

   GST leaf
   = hash(id, UST root)

2. Check if the epoch key is computed correctly epoch key == hash (id, epoch, nonce)

Unirep Protocol

Give reputation



User_1

epk_1

epk_1 = hash(id, epoch, nonce)

epk_1 proof

submit proof on chain

Unirep contract

attesterID = 1

reputation to epk_1 via an attester

User_2

Unirep Protocol

Users can verify the proof and send reputation to the epoch key

1. Users calls the smart contract through an ethereum wallet or a smart contract
2. The smart contract checks if the attester has signed up
3. Reputation hash chain is computed

```
hashChain[epochKey] = hash(reputation, hashChain[epochKey])
reputation = hash(attesterId, posRep, negRep,
                      graffiti, signUpFlag)
```

⇨ **Non-repudiable:** user can not omit any attestation

If user omits an attestation, the hashchain result will be different

Unirep Protocol

If a user spends his own reputation to give it to others

e.g. A user spends his 5 reputation to give 5 negative reputation to others

How to prevent an attester from double spending his reputation?

- **Idea: Proof of reputation nullifier**
  - A reputation nullifier:

    `hash(reputationDomain, identity, epoch ,nonce, attesterId)`
  - nonce < (posRep - negRep)
  - spends 5 reputations = submits 5 reputation nullifiers

Unirep Protocol

- Reputation nullifier example:

```
hash(reputation_domain, identity, epoch ,nonce, attester_id)
```

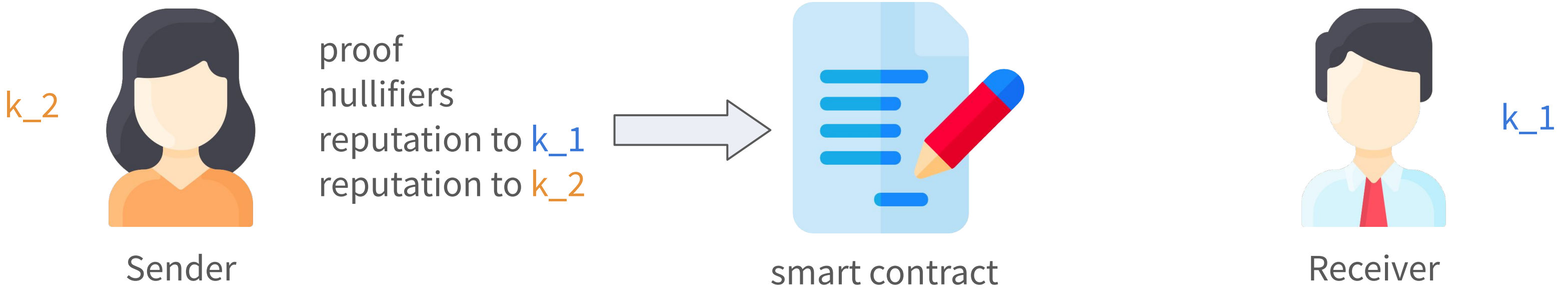
nonce = 0	4e07408562bed
nonce = 1	b8b60ce05c1de
nonce = 2	cfe3ad16b7223
nonce = 3	0967de01f640b
nonce = 4	b7e4729b49fce

Unirep Protocol

Sender sends reputation to attester the receiver's epoch key k_1

1. Sender generate a proof including n different nullifiers and his epoch key k_2
2. Sender submit the proof on smart contract and give reputation to k_1
3. Sender also give the same amount of negative reputation to k_2
4. If the proof is correct and the nullifiers are not seen before

   update the hashchain of k_1 and k_2



k_2

Sender

proof
nullifiers
reputation to k_1
reputation to k_2

smart contract

Receiver

k_1

Unirep Protocol

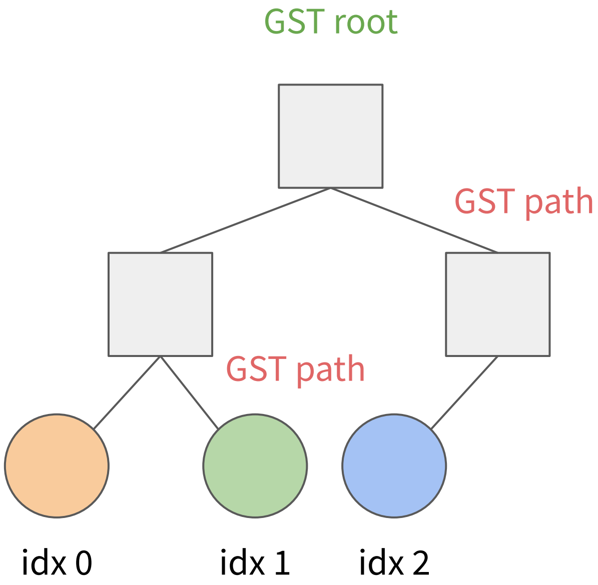
Reputation proving circuit:

- public input: epoch key ,epoch, GST root, # nullifiers
- public output: nullifiers
- private input:

  identity, epoch key nonce, UST root, UST path

  posRep, negRep, GST path, nullifier nonces

1. Check if user exists in the Global State Tree
2. Check correctness of epoch key to receive negRep
3. Check total reputation is greater than 0
4. Check nullifiers are valid



GST leaf
= hash(id, UST root)

epoch key == hash (id, epoch, nonce)
nullifier = hash (domain, id, epoch, nonce, att_id)
(posRep - negRep) > # nullifiers

Unirep Protocol

How can an attester airdrop users?

1. How can the attester make sure the user has signed up in the app before?

without revealing the user's identity and his previous epoch keys

2. How can the proof promise that one user only get one airdrop per epoch?

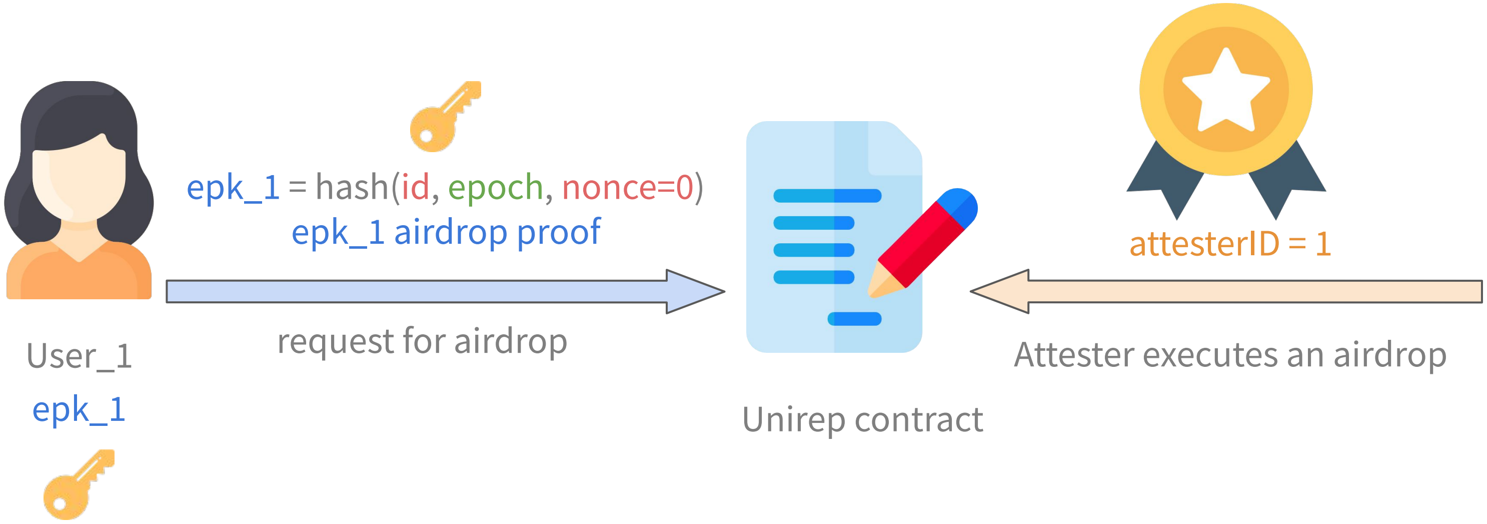
- **Idea: Proof of sign up in the specified application**
  - A sign up flag in reputation leaf:

    `hash(posRep, negRep, graffiti, signUpFlag)`
  - Specified epoch key

    `hash(identity, epoch ,nonce=0)`

Unirep Protocol

- **Give airdrop**



Unirep Protocol

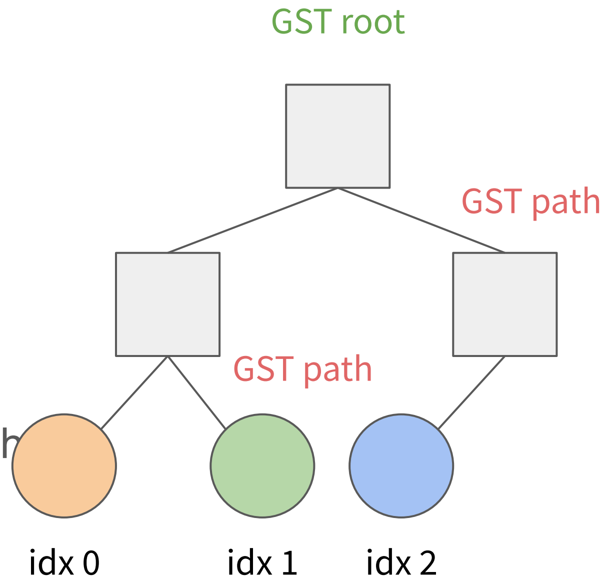
Sign up in app proving circuit:

- public input: epoch key ,epoch, GST root
- private input:

  identity, epoch key nonce, UST root, UST path

  sign up flag, GST path

1. Check if user exists in the Global State Tree
2. Check correctness of epoch key to receive negRep
3. Check if sign up flag is true

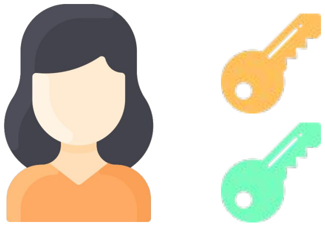


GST leaf = hash(id, UST root)

epoch key == hash (id, epoch, nonce=0)
signUpFlag == 1

Unirep Protocol

How can users receive all of reputation?



posRep = 5 from Airb*b

negRep = 3 from T*itter

- Reputations are sent to different epoch keys and the user state tree is not updated
- **Epoch transition**
  - Every `epochLength` seconds, one epoch ends and the next epoch begins
  - Sealed hashchains

```
hashChain[epochKey] = hash(1, hashChain[epochKey])
```

  - Sealed hash chain will be inserted into the epoch tree
  - The epoch key of sealed hash chain cannot receive reputation anymore

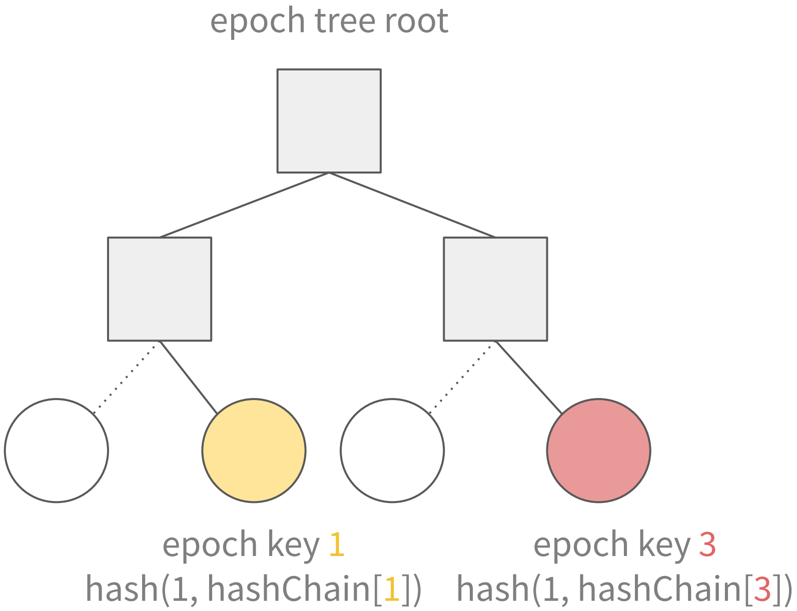
Unirep Protocol

Epoch tree

- *leaf index:* epoch key
- *leaf value:* sealed hashchain of the epoch key

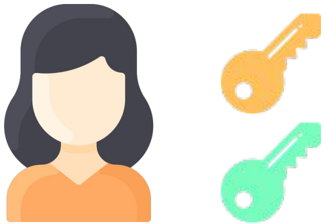
⇨ **Non-repudiable:** users cannot omit any epoch key

The circuit will process exactly *n* epoch keys, and all these *n* epoch keys should output the same epoch tree root.



```
hashChain[epochKey] = hash(rep_i, hashChain[epochKey])
hashChain[epochKey] = hash(rep_n, hash(rep_{n-1}, hash(....,hash(rep_1, 0))))
```

Unirep Protocol



posRep = 5 from Airb*b

negRep = 3 from T*itter

How can users receive all of reputation?

- ***User state transition*** **from epoch n to epoch m**
  - Check if user exists in the GST in epoch n

    `hash(id, oldUSTRoot)`
  - Process the attestations of the epoch keys and update UST
  - Compute and output new GST leaf in epoch m

    `hash(id, newUSTRoot)`

Unirep Protocol

User state transition proving circuit:

- public input: GST root, epoch tree root
- public output: new GST leaf, epoch key nullifiers
- private input:

  identity, UST roots, posRep, negRep, GST path,

  reputation, epoch tree paths, sealed hashchain

Unirep Protocol

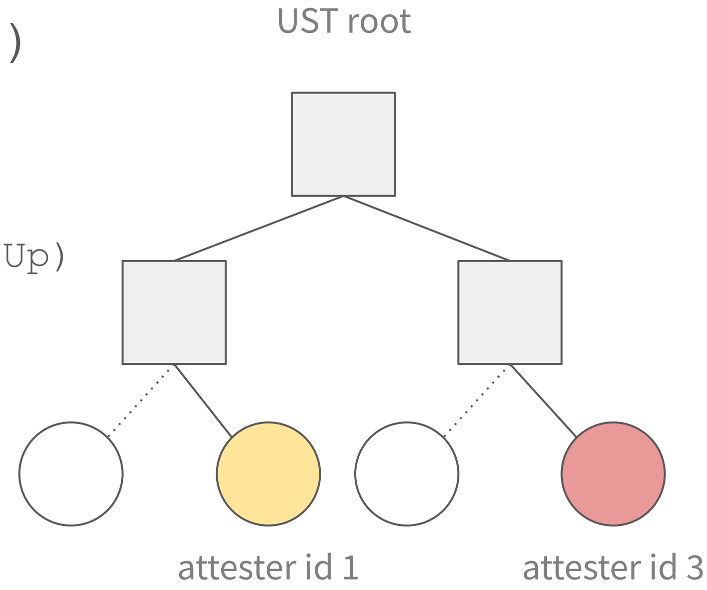
User state transition proving circuit includes:

- Compute k epoch keys using nonce from 0 to (k-1)
- Update UST
  1. Update hashed reputation:

     `hash(posRep, negRep, graffiti, signUp)`
  2. Compute new UST root

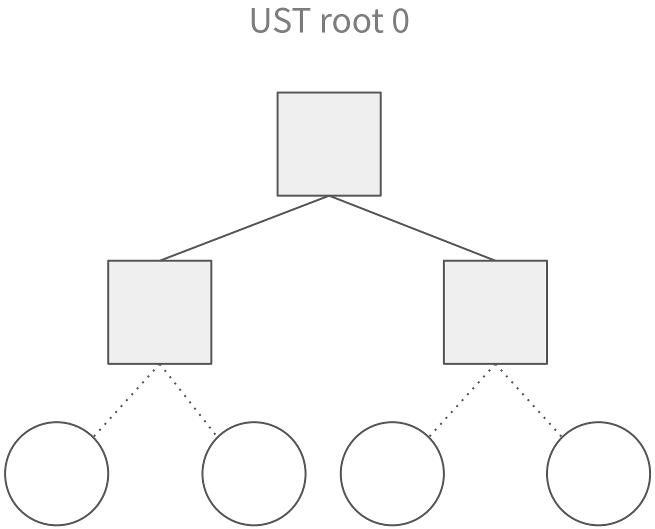
     n reputations ➡ (n+1) UST roots
  3. Compute hash chain results
- epoch keys & sealed hash chain

  matches epoch tree



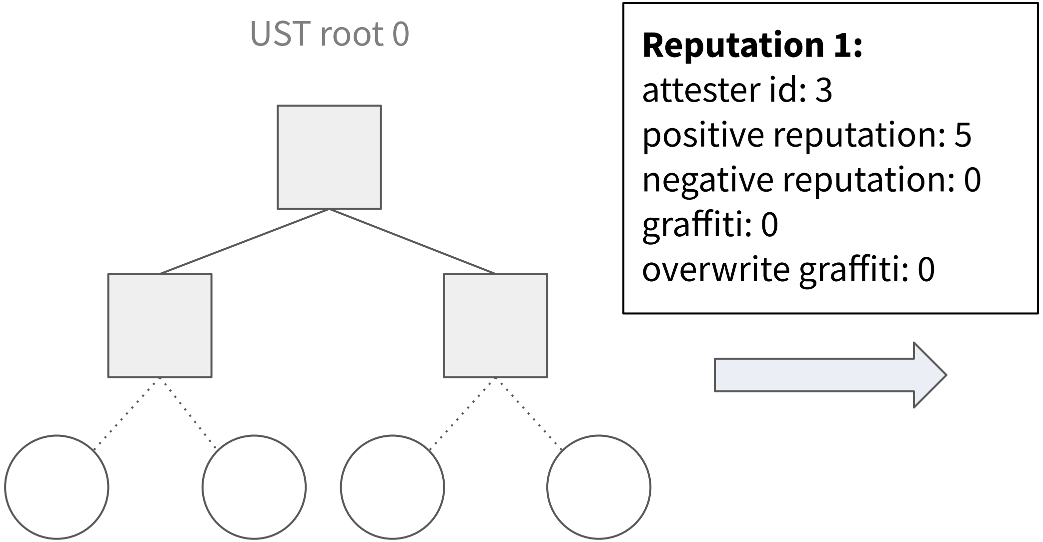
Unirep Protocol

User state transition proving circuit:



Unirep Protocol

User state transition proving circuit:



Reputation 1:
attester id: 3
positive reputation: 5
negative reputation: 0
graffiti: 0
overwrite graffiti: 0

Unirep Protocol

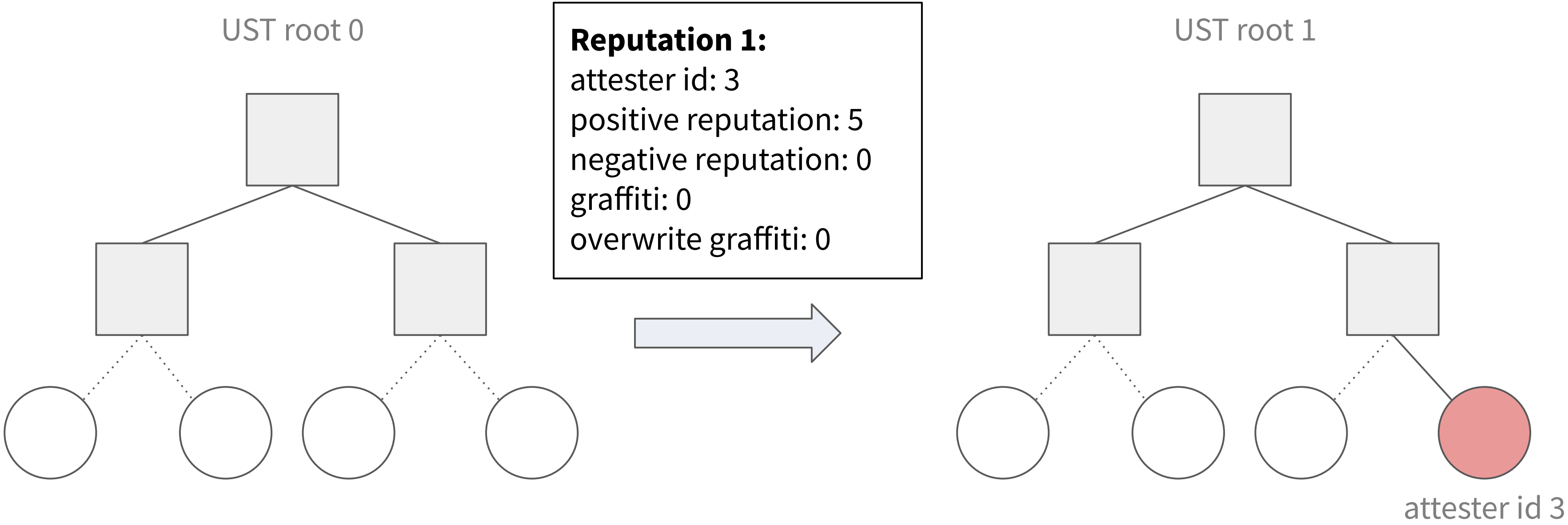
leaf_3 = hash(5, 0, 0)

hash_reputation_1 = hash(3,5,0,0,0)

hashchain_1 =

hash(hash_reputation_1, 0)

User state transition proving circuit:



Reputation 1:
attester id: 3
positive reputation: 5
negative reputation: 0
graffiti: 0
overwrite graffiti: 0

Unirep Protocol

User state transition proving circuit:

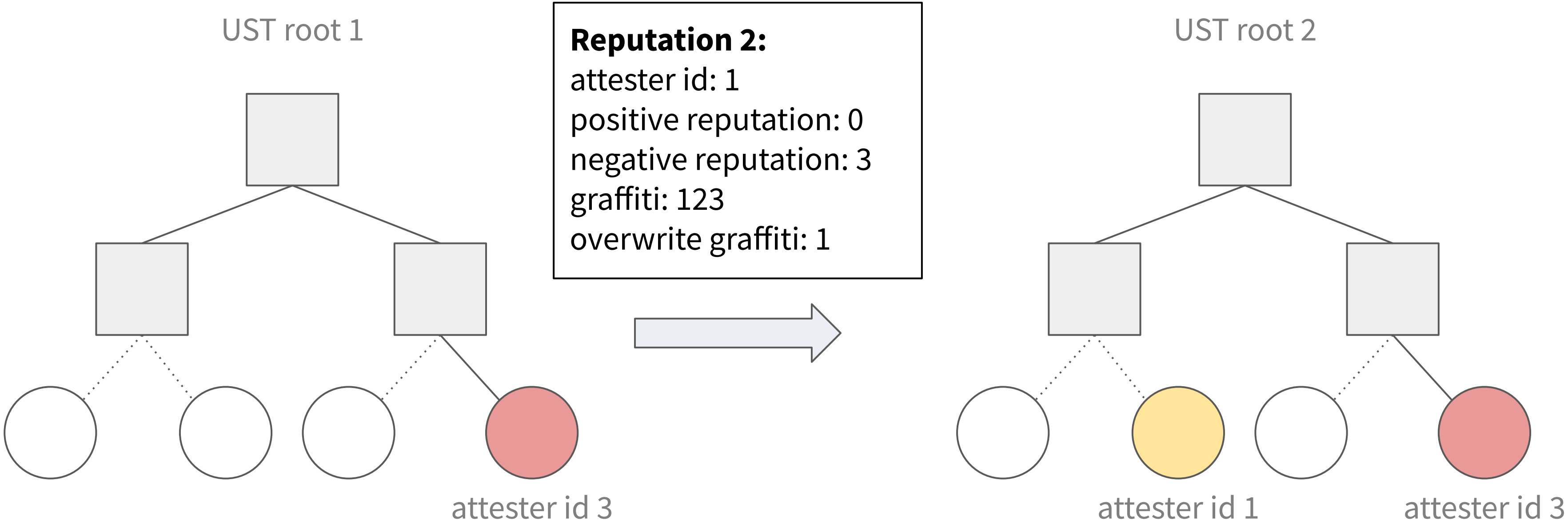
leaf_1 = hash(0, 3, 1)

hash_reputation_2 = hash(1,0,3,123,1)

hashchain_2 =

hash(hash_reputation_2, hashchain_1)

Reputation 2:
attester id: 1
positive reputation: 0
negative reputation: 3
graffiti: 123
overwrite graffiti: 1



Unirep Protocol

User state transition proving circuit:

Epoch keys:

```
hash(identity, epoch ,nonce)
```

identity + epoch = 1 +

nonce = 0 → 4e07408562bed

nonce = 1 → b8b60ce05c1de

nonce = 2 → cfe3ad16b7223

nonce = 3 → 0967de01f640b

nonce = 4 → b7e4729b49fce

Unirep Protocol

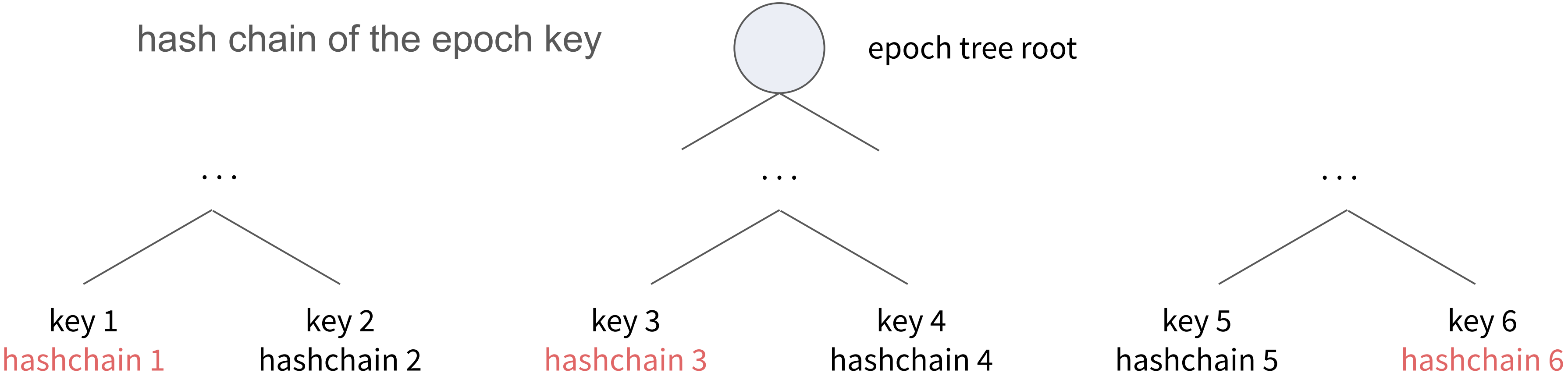
User state transition proving circuit:

- Epoch tree:

  Leaf ID: epoch key

  Leaf value:

  hash chain of the epoch key



Unirep Protocol

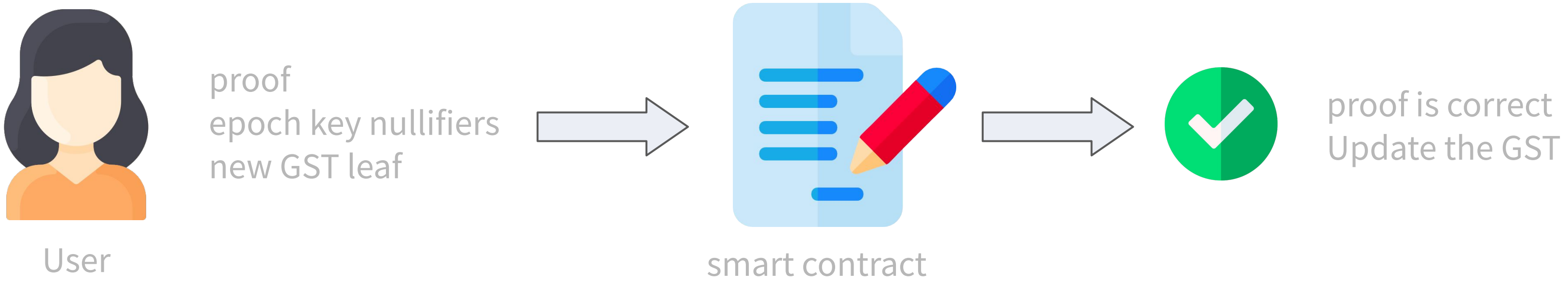
User state transition proving circuit:

- Compute new GST leaf
- `hash(id, newUSTRoot)`

Unirep Protocol

User state transition proving circuit:

- Compute epoch key nullifiers:
  1. Prevent users or others from re-using epoch keys to receive reputation
  2. Prevent double user state transition
- `epochKeyNullifier = hash(epkDomain, id, fromEpoch, nonce)`



User

proof
epoch key nullifiers
new GST leaf

smart contract

proof is correct
Update the GST

Unirep Protocol

After user state transition,

user can voluntarily prove how much reputation he has:

- User may prove
  - The reputation >= `provingRep`
  - What is the pre-image of a graffiti
  - User has a sign up flag

  from an attester id j

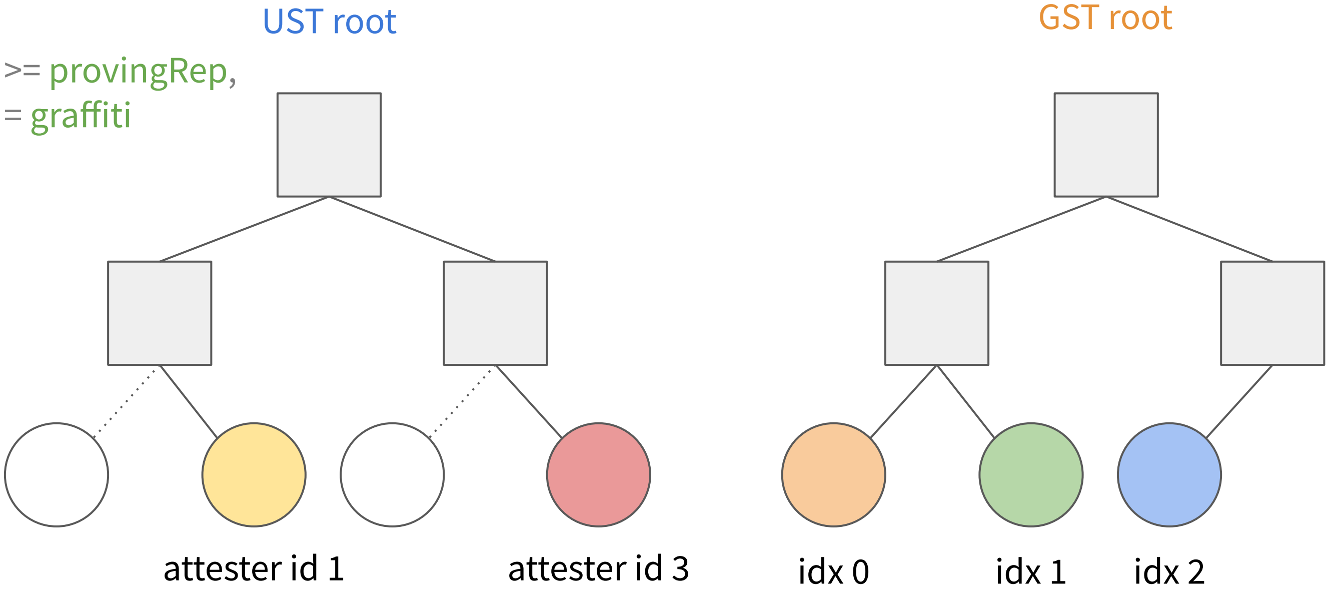
- Idea: prove a leaf of the UST, and the GST leaf existed in the current epoch

```
USTLeaf = hash(posRep, negRep, graffiti, signUpFlag)
GSTLeaf = hash(id, USTRoot)
```

Unirep Protocol

User can voluntarily prove how much reputation he has:

(posRep - negRep) >= provingRep,
hash(pre-image) = = graffiti
signUpFlag == 1

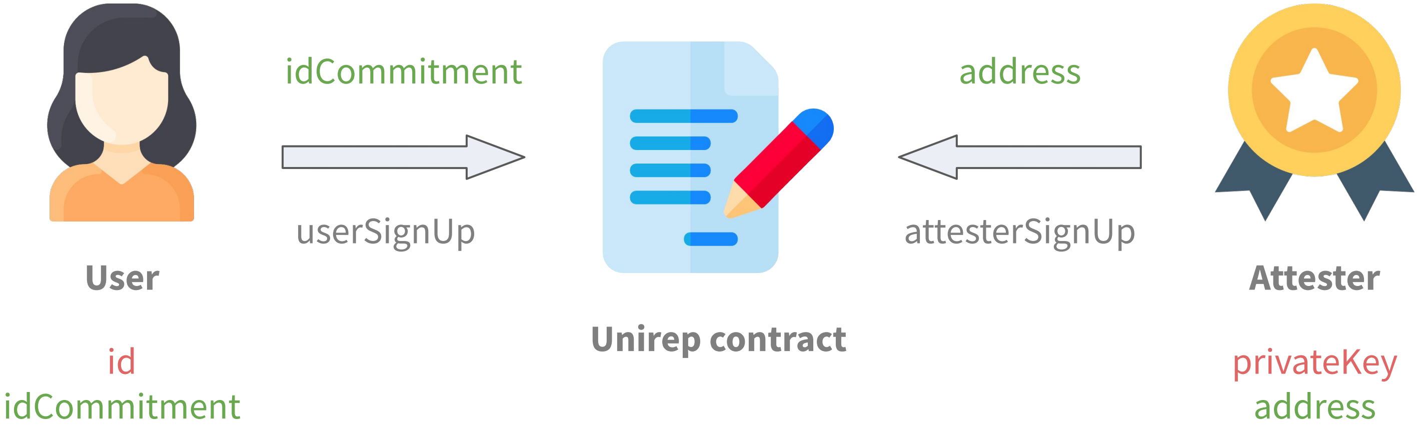


USTLeaf = hash(posRep, negRep, graffiti, signUpFlag)

GST leaf = hash(id, USTRoot)

Unirep Protocol Overview

User/ attester signs up



idCommitment

userSignUp

address

attesterSignUp

User

id
idCommitment

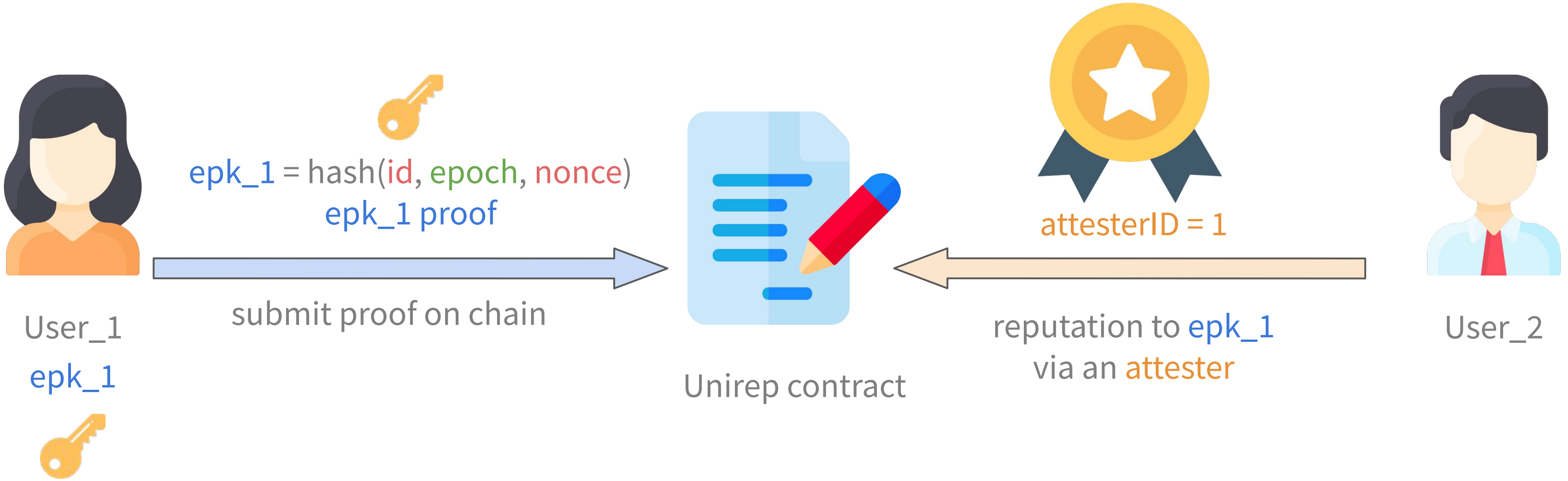
Unirep contract

Attester

privateKey
address

Unirep Protocol Overview

Give reputation



epk_1 = hash(id, epoch, nonce)

epk_1 proof

submit proof on chain

User_1

epk_1

Unirep contract

attesterID = 1

reputation to epk_1

via an attester

User_2

Unirep Protocol Overview

Epoch transition

Unirep contract
epoch += 1

Seal hash chains of epoch keys

hash(1, hashchain_1)

hash(1, hashchain_2)

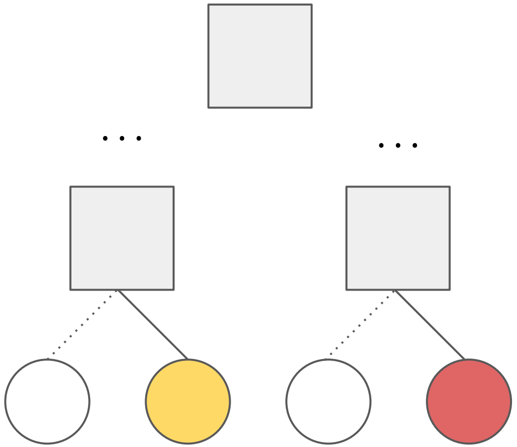
hash(1, hashchain_3)

hash(1, hashchain_4)

hash(1, hashchain_5)

Build epoch tree

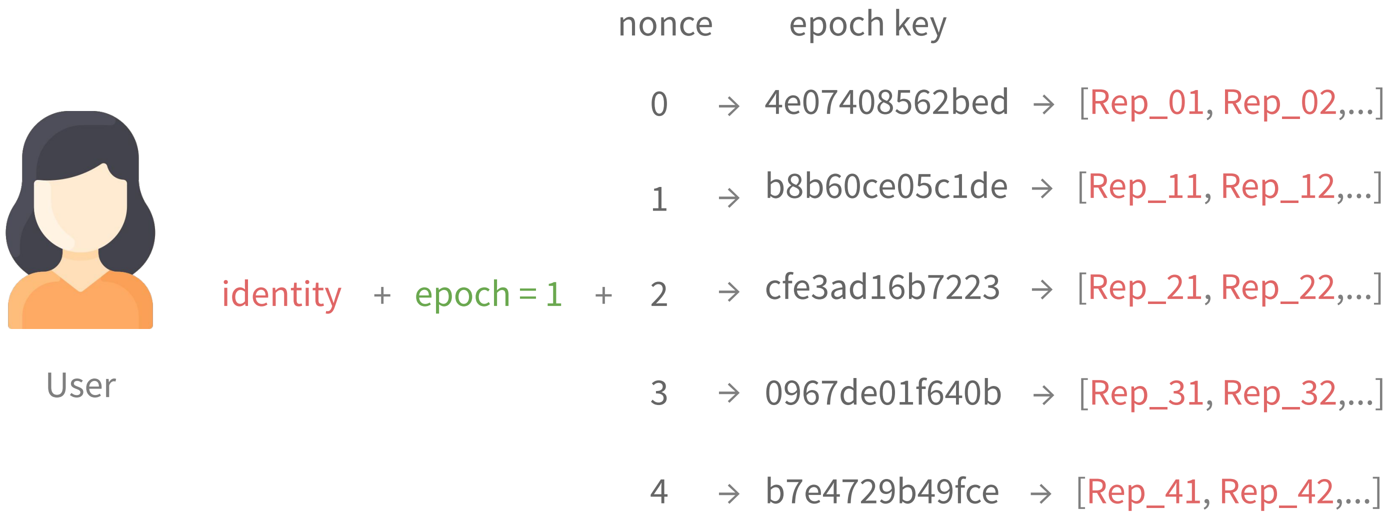
epoch tree root



hashchain_3 hashchain_1

Unirep Protocol Overview

Receive reputation (User State Transition)



nonce epoch key

identity + epoch = 1 +

0 → 4e07408562bed → [Rep_01, Rep_02,...]

1 → b8b60ce05c1de → [Rep_11, Rep_12,...]

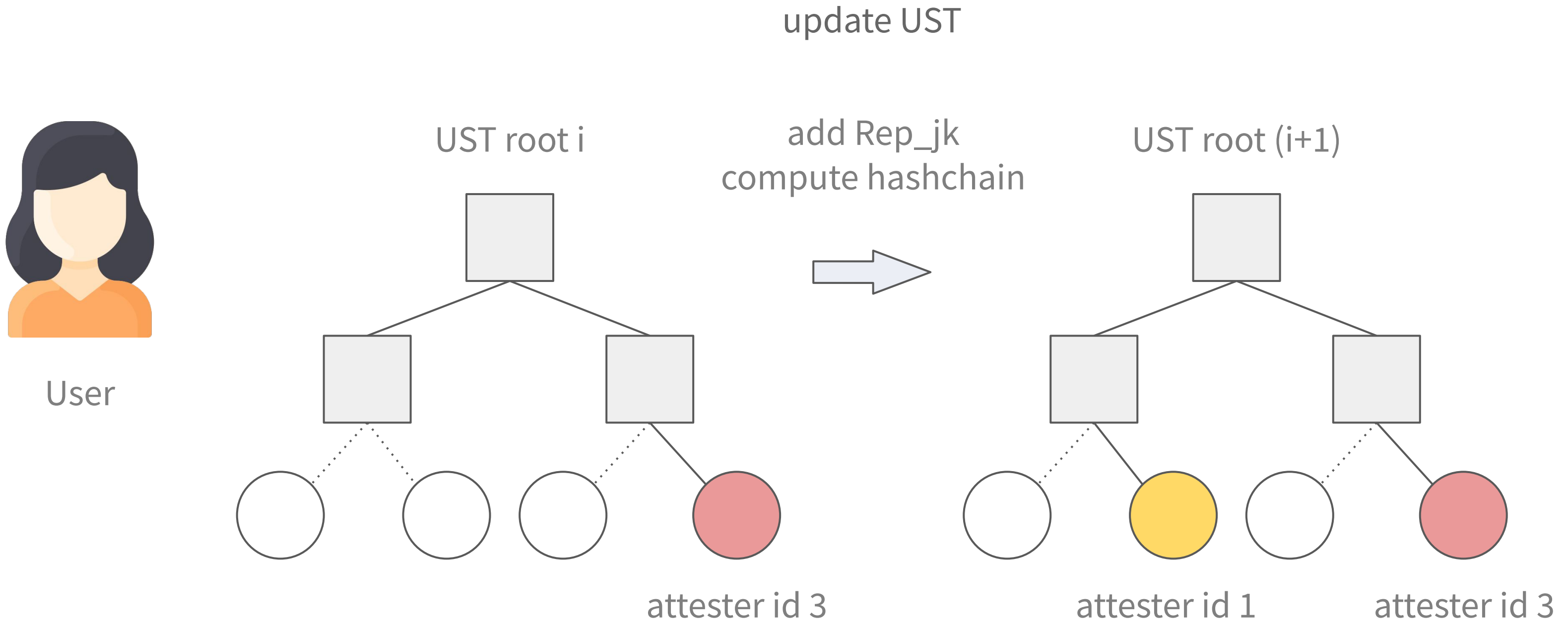
2 → cfe3ad16b7223 → [Rep_21, Rep_22,...]

3 → 0967de01f640b → [Rep_31, Rep_32,...]

4 → b7e4729b49fce → [Rep_41, Rep_42,...]

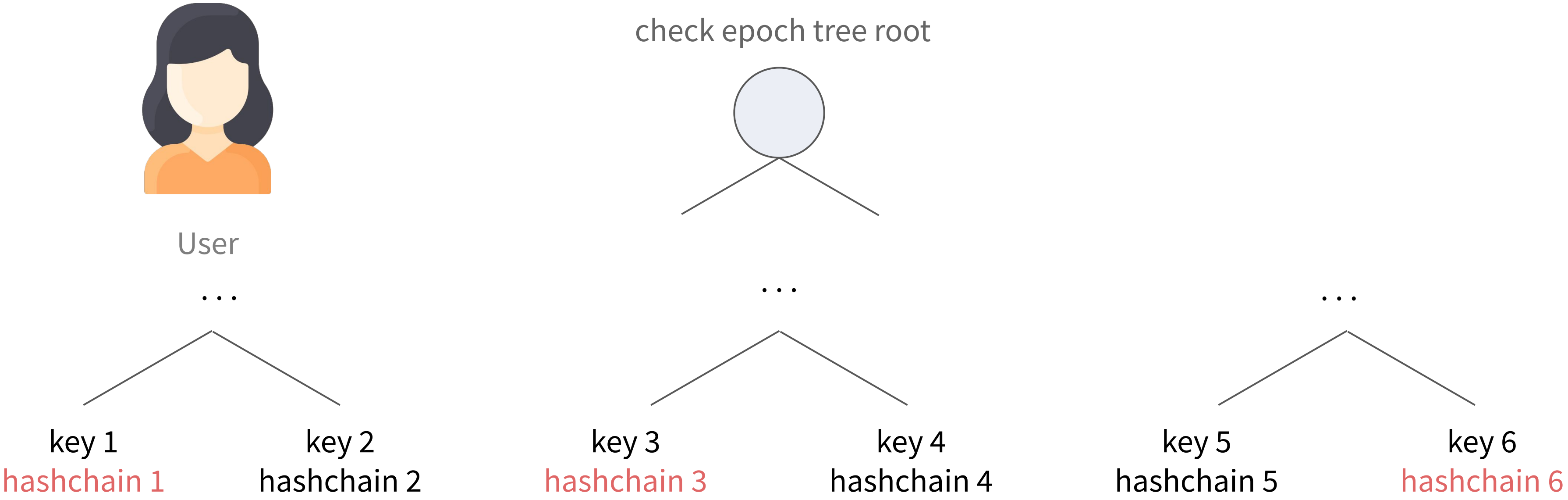
Unirep Protocol Overview

Receive reputation (User State Transition)



Unirep Protocol Overview

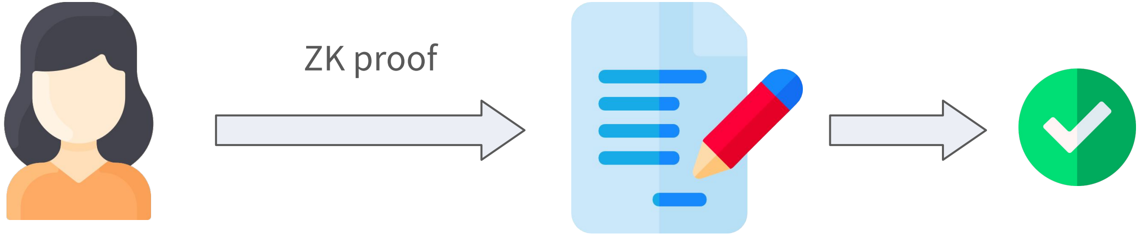
Receive reputation (User State Transition)



Unirep Protocol Overview

Receive reputation

Computes a new reputation state
Generates a ZK proof



User

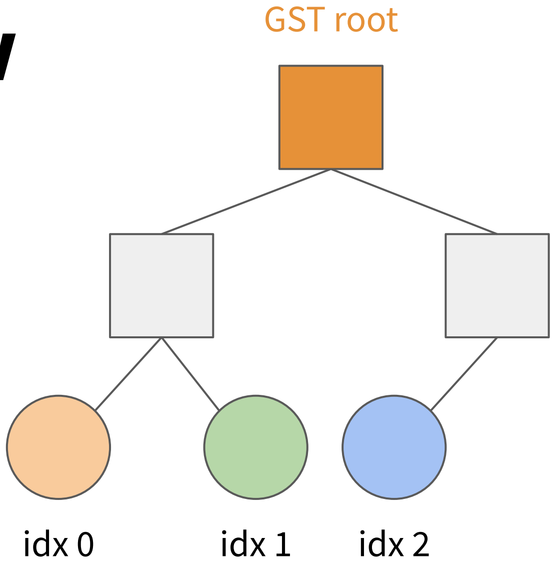
ZK proof

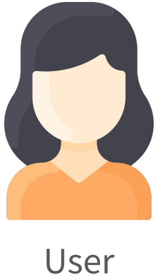
smart contract

Proof is correct
Updates Unirep state

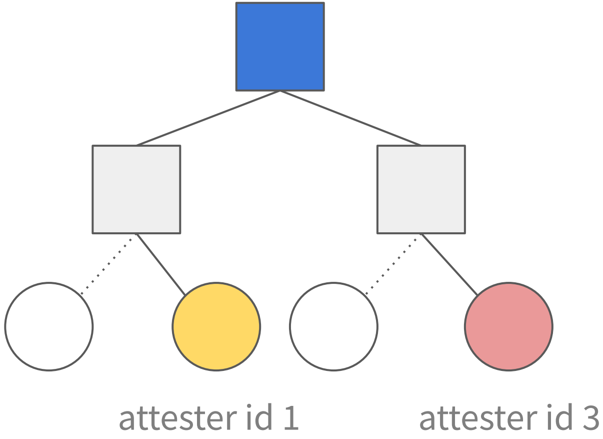
Unirep Protocol Overview

Prove reputation





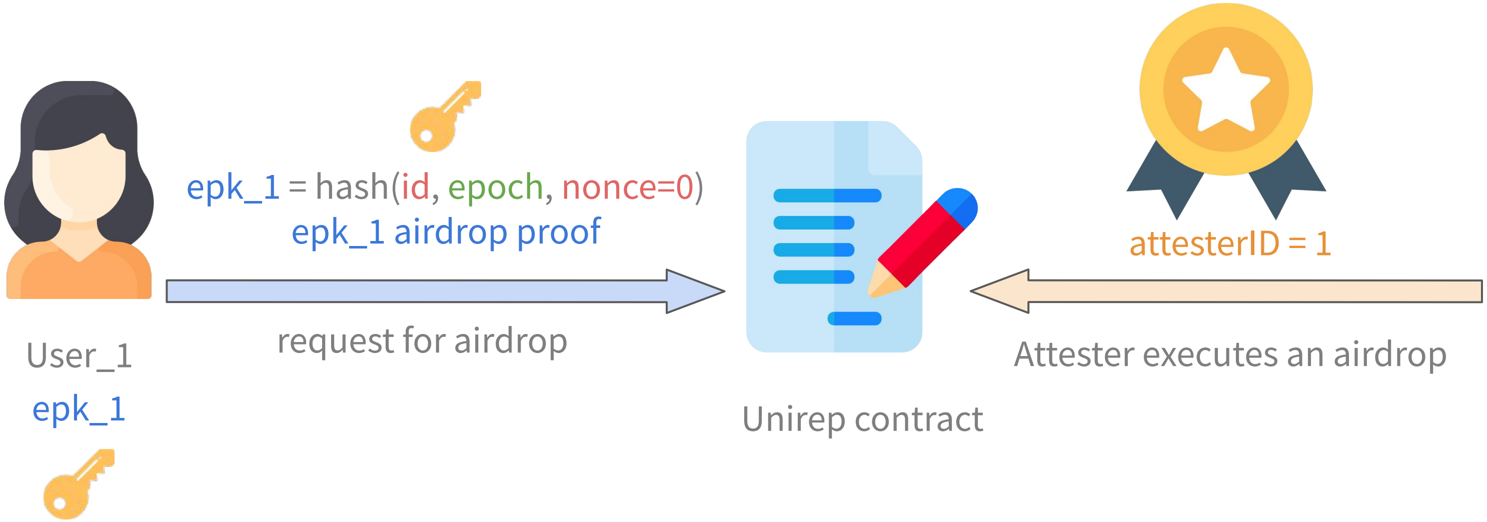
hash(id, UST root) = one of GST leaves



(posRep - negRep) >= provingRep,
hash(pre-image) = = graffiti
signUpFlag == 1

Unirep Protocol Overview

- **Give airdrop**


epk_1 = hash(id, epoch, nonce=0)
epk_1 airdrop proof
attesterID = 1
request for airdrop
Attester executes an airdrop
User_1
epk_1
Unirep contract