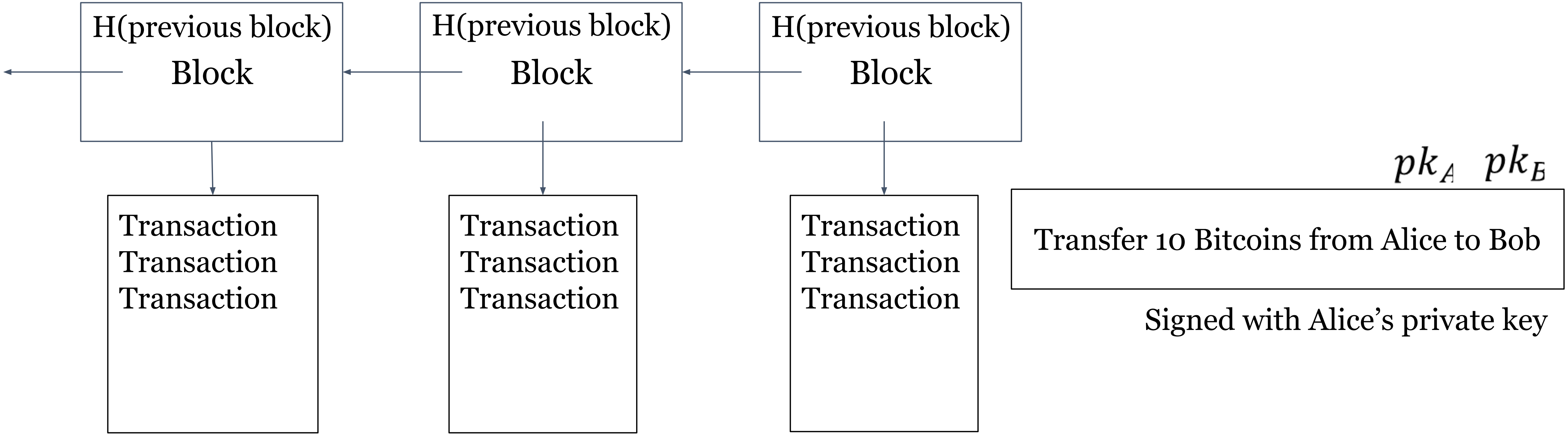
Privacy-preserving Crypto-currencies

Blockchain



- Append-only authenticated list
- A random party is selected to propose the next block (mining)
- Everyone checks the data of the new block is **valid**
- More than 50% honest parties → consensus without a centralized trusted party

Account model vs transaction model

time

Create 25 coins and credit to Alice$_{\text{ASSERTED BY MINERS}}$

Transfer 17 coins from Alice to Bob$_{\text{SIGNED(Alice)}}$

Transfer 8 coins from Bob to Carol$_{\text{SIGNED(Bob)}}$

Transfer 5 coins from Carol to Alice$_{\text{SIGNED(Carol)}}$

Transfer 15 coins from Alice to David$_{\text{SIGNED(Alice)}}$

is this valid?

SIMPLIFICATION: only one transaction per block

A transaction-based ledger (Bitcoin)

time

1
Inputs: Ø
Outputs: 25.0→Alice

2
Inputs: 1[0]
Outputs: 17.0→Bob, 8.0→Alice
SIGNED(Alice)

change address

3
Inputs: 2[0]
Outputs: 8.0→Carol, 9.0→Bob
SIGNED(Bob)

4
Inputs: 2[1]
Outputs: 6.0→David, 2.0→Alice
SIGNED(Alice)

is this valid?

SIMPLIFICATION: only one transaction per block

Meta data for efficient validation

- Account model: account balances

- Transaction model: UTXO (unspent output of transactions)

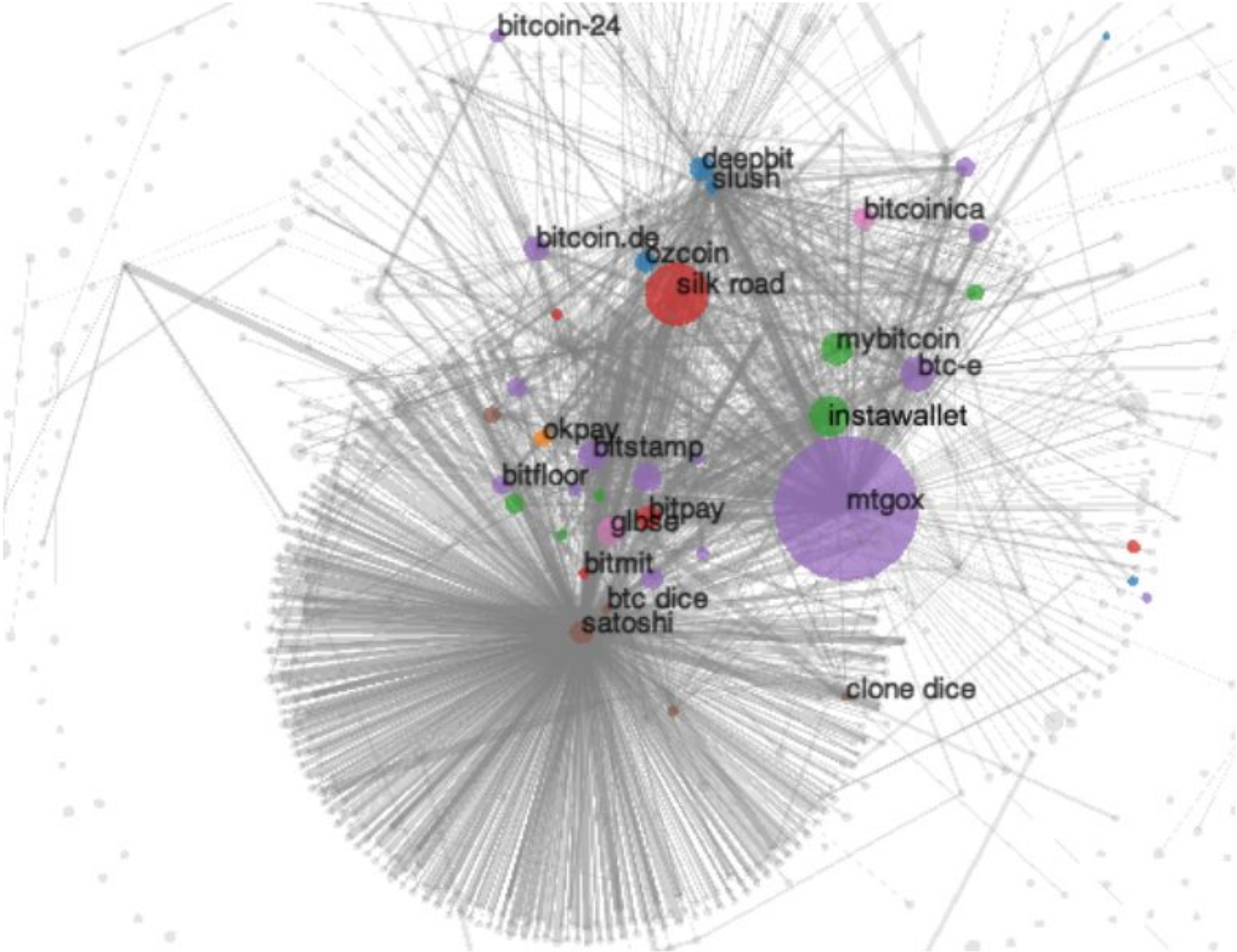
  - A set of unspent transactions
  - Each new transaction destroys 1 (or several) elements in the set, and insert 2 elements into the set

Privacy of Bitcoin

- ✔ IDs are random public keys
- ✔ Permissionless: one can create many accounts/IDs

- × IDs are deterministic
- × Transactions are posted publicly on the blockchain

Linkage attack on Bitcoin network



A fistful of bitcoins: characterizing payments among men with no names, Meiklejohn et al. 2013

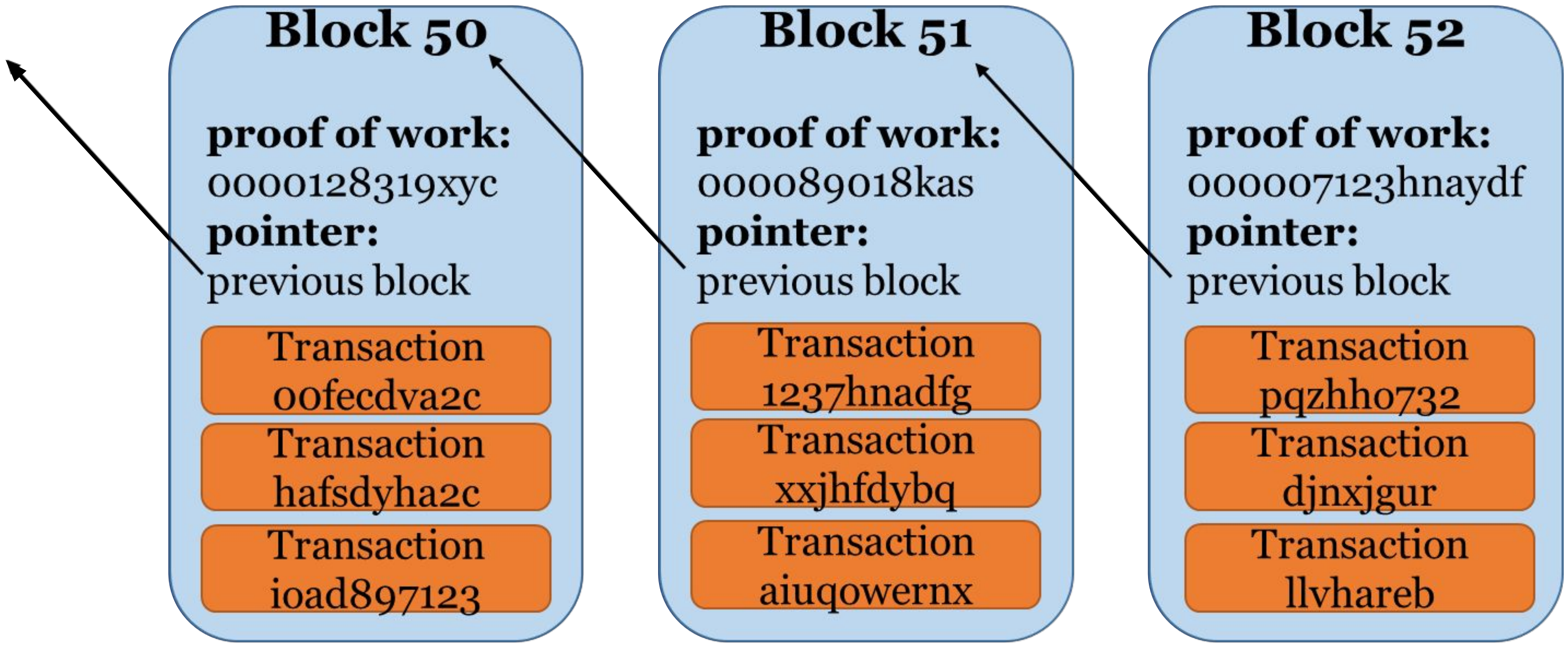
Privacy problems in blockchain

All data are posted publicly on the chain for users to validate

Pseudo-anonymity

Can we prove data validity without revealing the data

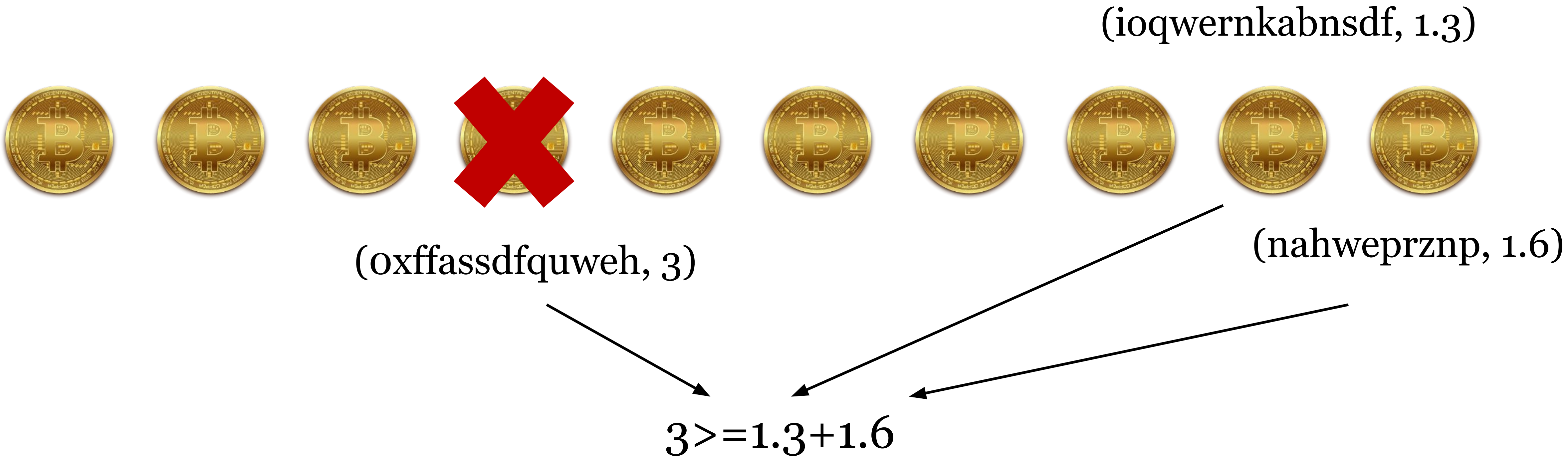
Solution: zero knowledge proof



Publish **zero knowledge proofs** of data validity on blockchain

Zerocash/Zcash

UTXO model (unspent transaction output)



Commitments and set membership

H() H() H() H() H() H()

H(0xffassdfquweh, 3)

Type equation here.

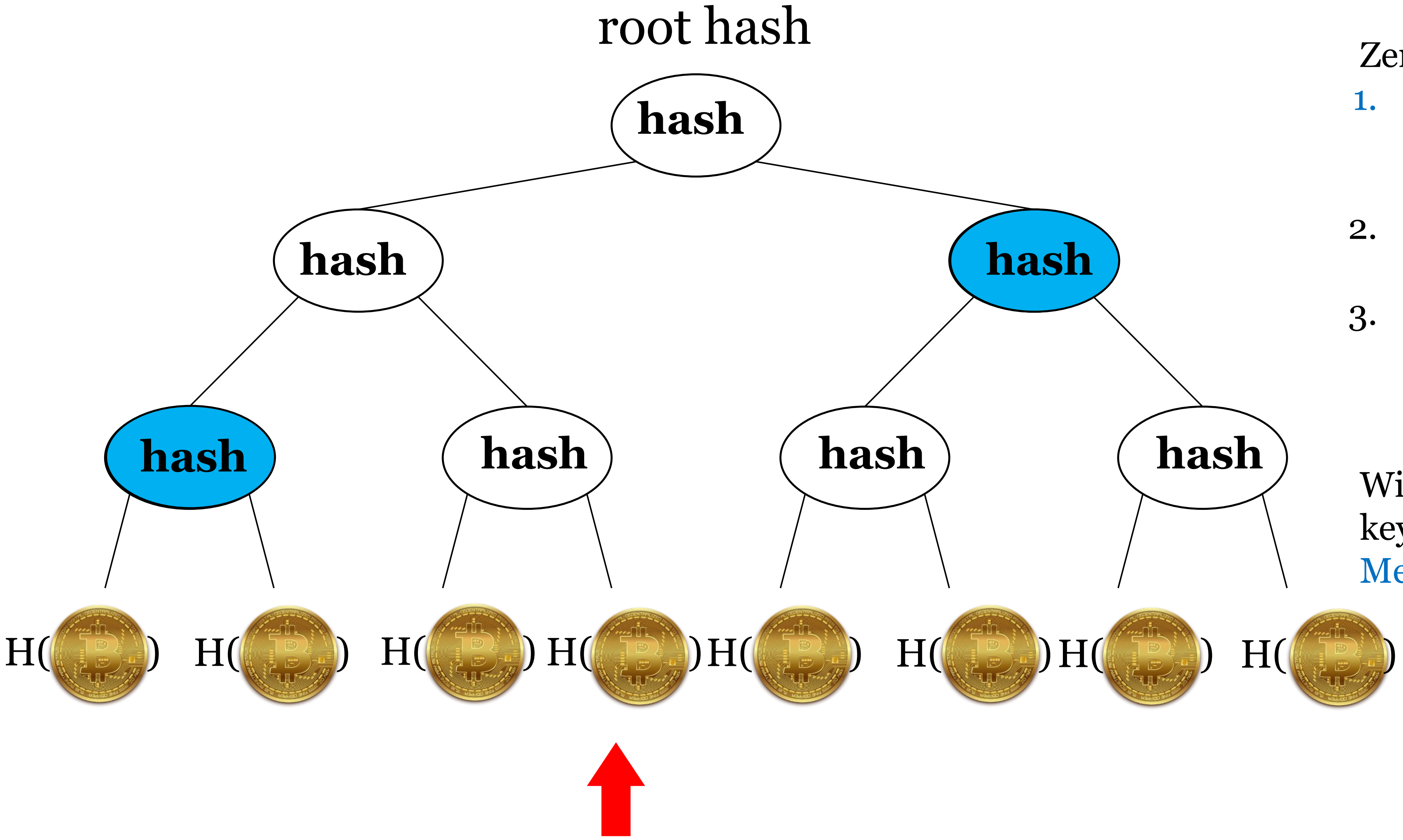
Zero knowledge proof:

1. The coin/commitment/hash is in the set
2. I know the pre-image of the commitment/hash
3. I know the secret key of the public key

Witness/secret of ZKP: secret key, public key, amount

Problem: the function of ZKP is linear

Proving membership: Merkle hash tree

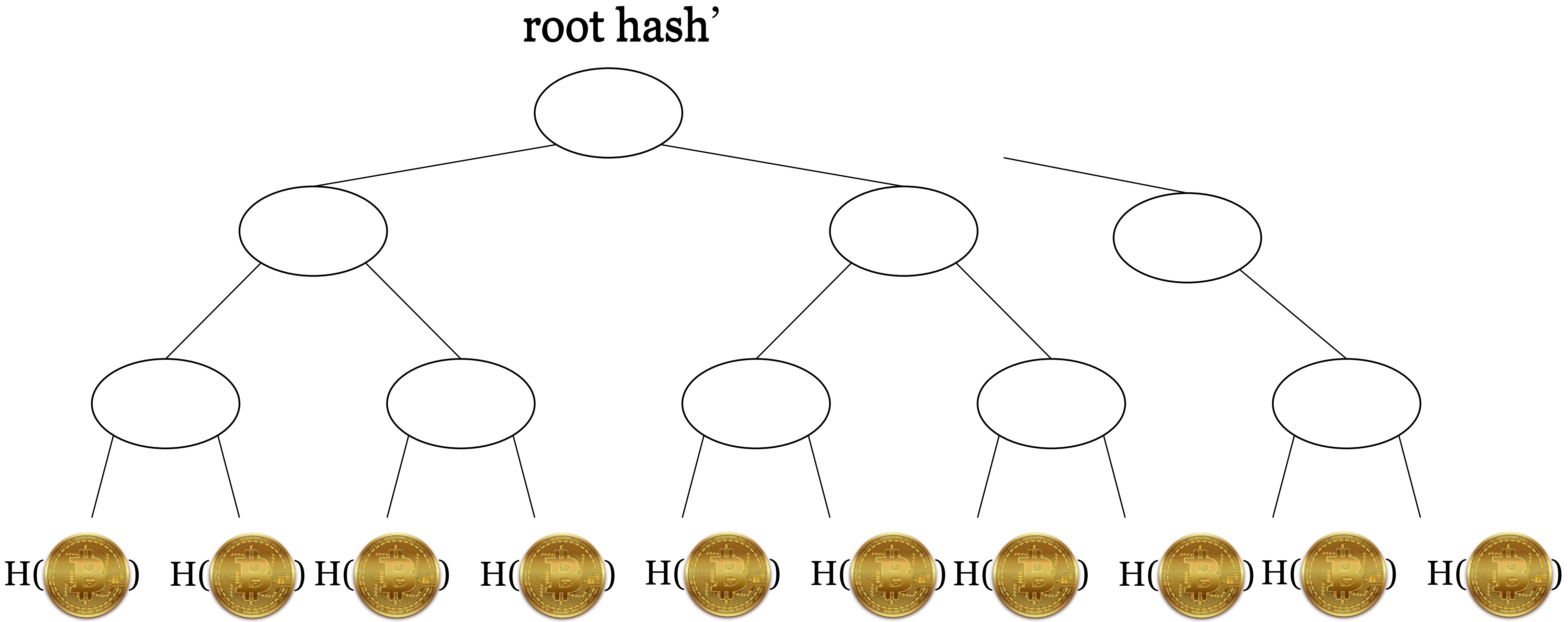


Zero knowledge proof:

1. There is a Merkle tree path for the coin/commitment/hash
2. I know the pre-image of the commitment/hash
3. I know the secret key of the public key

Witness/secret of ZKP: secret key, public key, amount, Merkle tree path

Generating new coins



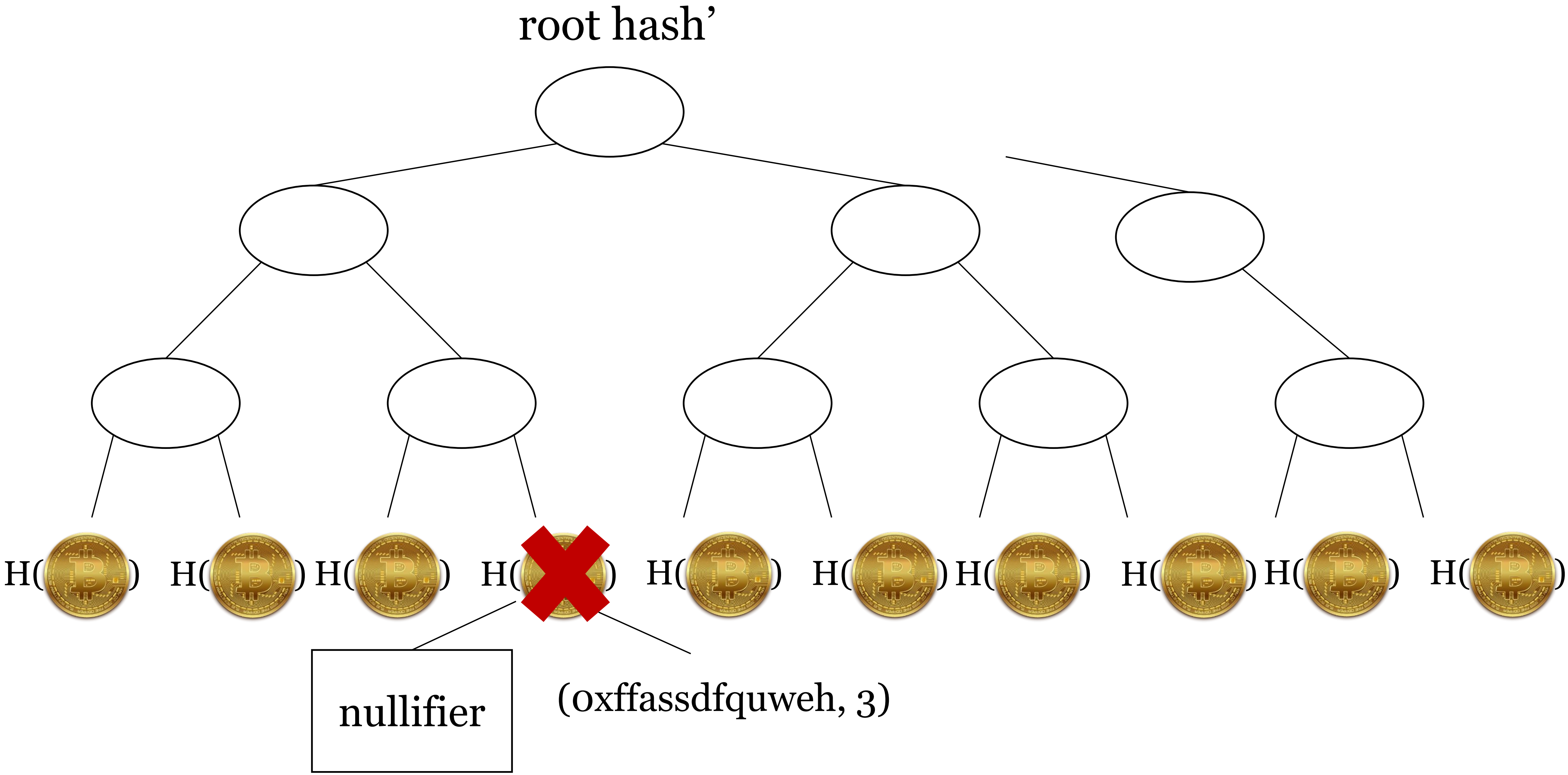
New coins and new commitments

Zero knowledge proof:

1. There is a Merkle tree path for the coin/commitment/hash
2. I know the pre-image of the commitment/hash
3. I know the secret key of the public key
4. The amount of two new coins are less than the old coin
5. The commitments are computed from the new coins

Witness/secret of ZKP: secret key, public key, amount, Merkle tree path, new coins (receiver public key, amount)

Preventing double spending: nullifier

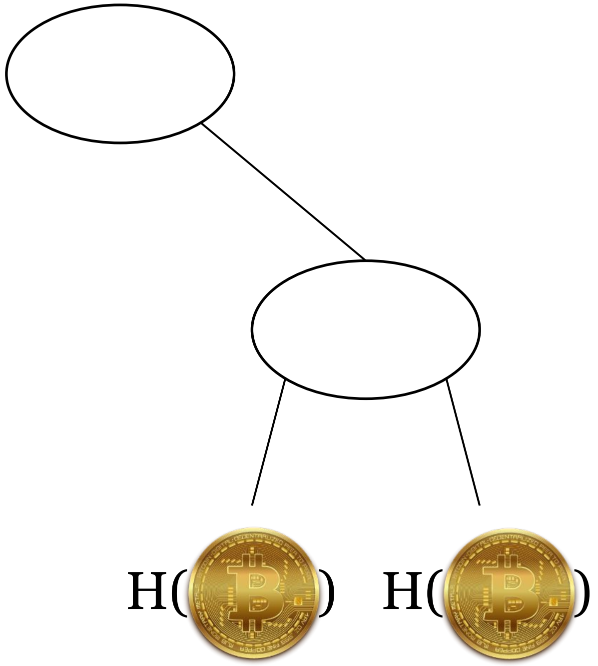


Sending money

- Encrypt under receiver's public key

Zero knowledge proof:

1. There is a Merkle tree path for the coin/commitment/hash
2. I know the pre-image of the commitment/hash
3. I know the secret key of the public key
4. The amount of two new coins are less than the old coin
5. The commitments are computed from the new coins
6. The encryption is the new coin, etc.



Zcash

- Uses zero knowledge proof
- Avoid linear scan: Merkle tree
- Double spending: nullifier
- Send money: encryption and signature

Zero knowledge proof scheme: zkSNARK

✔ Supports all functions (modeled as arithmetic circuit)
✔ Constant proof size < 200 bytes
✔ Fast verification time 3ms

× Function dependent **trusted setup**
× Slow prover time (modular exponentiations for every gate)